Hotel Reservation Software

Group 1 Project Documentation, Rev 0

CMSC 495 ( 6380) Current Trends and Projects in Computer Science (2205)

University of Maryland Global Campus

Summer 2020

Professor Hung Dao

July 12th, 2020

Group 1 Members:

Jacob Valentine

Leonardo Elias

Emmanuel Girin

Marques Young

# Revision Table

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Description | Contributor |
| 0 | 7/11/2020 | Merged the project documentation into a central document. Formatted the document to fit. Revised the requirements section. | Jacob Valentine |
| 1 | 7/12/2020 | Expanded the User Guide section to include installation instructions, system requirements, and step by step instructions for operation. | Marques Young,  Jacob Valentine,  Emmanuel Girin,  and Leonardo Elias |
| 2 | 7/12/2020 | Applied and verified the actual result and pass/fail states for each test case in the Test Case section. | Leonardo Elias and Emmanuel Girin |

# Table of Contents

# Project Plan

## Revision Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Description** | **Contributor** |
| 0 | 06/01/2020 | Creating Document, Cover Page, Revision Table | Jacob Valentine |
| 1 | 06/01/2020 | Formatting Update, Adding Sections, Updating Timeline | Emmanuel Girin |
| 2 | 06/01/2020 | Filled in Requirement Specifications / System Specifications | Jacob Valentine |
| 3 | 06/01/2020 | Added Software Management Section, general formatting, & spelling | Emmanuel Girin |
| 4 | 06/02/2020 | Reviewed and discussed | Leonardo Elias |
| 5 | 07/12/2020 | Revised the documentation to match the finished project. | Jacob Valentine |
| 6 | 7/13/2020 | Review / Update requirements | Emmanuel GIrin |
|  |  |  |  |

## Requirement Specifications

This project will involve a java applications which will allow for hotel staff to monitor, edit, and control information related to the allocation and specifications of the rooms of their hotel, including prices, reservation information, and reservation duration. The application will create a table database in which hotel room, reservation, customer, and user information will be securely stored, accessed, and edit through the application. A detailed list of application requirements will be provided in the “Project Requirements” section of this document.

## System Specification

### Development Platform

These requirements are specifically for the development of the application, and will not be necessarily required to run the finished application.

* + Operating System: Windows 7 or higher.
  + Processor: Intel Core i5 or equivalent
  + Memory: 4 GB
  + Disk Space: 5 GB
  + Software Resources: Java JRE 8, MySQL
  + Software Development Platform: NetBeans IDE 8.1 (64-bit)

### Operating Platform

These requirements are specifically for the platform that will run the finished applications.

* + Operating System: Windows Vista SP2
  + Processor: Pentium 2 266 MHz processor or equivalent
  + Memory: 3 MB of free RAM
  + Disk Space: 200 MB
  + Software Resources: Java JRE 8, MySQL

## Software Management

This project is hosted in a GitHub repository at <https://github.com/JacobValent25/CMSC_Hotel>. Each member of the group has been given access, and a revision document will be included in the repository to track changes and versions.

## Project Schedule

The project schedule follows the main points highlighted by the syllabus. The group chose to work together on the main parts of the project. As the project develops further splitting of tasks will occur. All changes will be reflected on this document

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task** | **Duration** | **Start** | **End** | **Personnel** |
| 1. Brainstorm Ideas Decide on Project | 5 Days | 05/22 | 05/27 | Jacob, Emmanuel, Leonardo (everyone) |
| 1. Project Requirements    1. Writing    2. Self-Review    3. Revise | 7 days  6 days  1 day  1 day | 05/27  05/27  06/02  06/02 | 06/02  06/01  06/02  06/02 | Everyone  Everyone  Everyone  Everyone |
| 1. Project Analysis    1. Analysis    2. Self-Review    3. Revise Document submission | 7 days  4 days  2 days  1 day | 06/02  06/02  06/06  06/08 | 06/09  06/05  06/08  06/09 | Everyone  Everyone  Everyone  Everyone |
| 1. Project Design | 7 Days | 06/10 | 06/16 | Everyone |
| 1. Project Test Plan and ICD | 7 Days | 06/17 | 06/23 | Everyone |
| 1. Implementation and Testing (Sprint 1 & 2)    1. Code    2. Test    3. User Guide | 14 Days  10 Days  7 Days  7 Days | 06/24  06/24  07/01  07/01 | 07/07  07/04  07/07  07/07 | Everyone  Everyone  Everyone  Emmanuel / Jacob |
| 1. Final Deliveries (Code, Binaries, Test Data, User’s Guide) | 5 Days | 07/08 | 07/12 | Everyone |

# Project Requirements

## Revision Table

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Description | Contributor |
| 0 | 06/01/2020 | Document Created with Cover Page, Revision Table | Jacob Valentine |
| 1 | 06/01/2020 | Formatting Update, Add Requirements Section. Added 24 Requirements | Emmanuel |
| 2 | 06/01/2020 | Filling in Topic Section | Emmanuel |
| 3 | 06/02/2020 | A refinement of the requirements. | Leonardo Elias |
| 4 | 06/07/2020 | Updated Requirements Per Feedback from Professor | Emmanuel |
| 5 | 07/11/202 | Revised Requirement table to align with the final project | Jacob Valentine |
| 6 | 7/13/2020 | Revised some Requirements to OPTIONAL | Emmanuel Girin |
|  |  |  |  |

## Topic

This software allows users to create a hotel reservation. The application is geared towards use by hotel front desk and sales employees. The software provides a solution for reserving available rooms on specific dates for customers. The focus will be on creating a java application with a easy to use functional UI for employees on the front end, which stores that information in an SQL Database. Employees of the hotel will be able to tie existing or new customers to reservations. Employees will also be able to change and delete existing reservations. The application will focus on maintaining strict security protocols for data privacy. It will also focus on retaining as much customer information as possible, so that customer habits can be tracked for marketing purposes.

## Requirements

All project requirements are saved in a OneDrive file, so that developers may collaborate. As you can see some of the requirements have been updated to be OPTIONAL, as they required more coding then we were prepared for. The space for implementation is there, but they need implementation at the installation of the program, so it would be specific to different builds. Setting up the different tiers of database users at the time of database creation for a client would be ideal. These matters will be left in the hands of the clients.

|  |  |
| --- | --- |
| # | Requirement |
| 1 | Software shall store personal consumer information (first and last name, phone #, address, email) |
| 2 | Software shall protect access to consumer information through username and password access |
| 3 | Software shall use SQL database to store consumer information |
| 4 | SQL Database shall contain three tiers of users **OPTIONAL Not implemented** |
| 5 | SQL Database shall have a master user that has super access to pass off to IT Team **IMPLEMENTED IN SQL MAKE FILE** |
| 6 | SQL Database shall have an mid-level user that can view and edit reservations / customer data **OPTIONAL NOT implemented** |
| 7 | SQL Database shall have a base-level user that can only view reservations (or maybe mid-level can delete reservation like manager access, and employee base level can only view and make reservations) **OPTIONAL Not Implemented** |
| 8 | Software shall allow a hotel staff to make a reservation for a guest |
| 9 | Software shall store reservations for rooms on specific dates |
| 10 | Software shall display available rooms on dates |
| 11 | Software shall have a basic UI with Menu |
| 12 | Menu shall display option for Finding Existing Reservation, Making new Reservation, deleting reservation and updating reservation. |
| 13 | Software shall require customer information to be captured when making a new reservation |
| 14 | Software shall ask staff it the customer already exists, and perform a look up by name, phone number, or email address |
| 15 | Software shall give Staff option to capture new customer Data if customer does not exist in the database (Register a new customer) |
| 16 | Existing reservations shall be performed by a customer look up |
| 17 | Software shall give staff the ability to change a reservation to a new date, room, and location (update reservation) - Reservation Updated by deleting and creating new |
| 18 | Software shall give staff ability to delete reservation |
| 19 | To Update or Create new Reservations: Staff shall enter date, which shall return a list of available room and descriptions |
| 20 | Rooms shall be organized by size (single, double, king, suite) |
| 21 | UI shall be simple to use Graphics UI with text boxes to enter in data and buttons to navigate and perform lookup functions |
| 22 | Software shall ask for Credential Logon at beginning of each use to ensure security |

# Project Analysis

Revision Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Description** | **Contributor** |
| 0 | 06/06/2020 | Creating Document, Cover Page, Revision Table | Jacob Valentine |
| 1 | 06/08/2020 | Editing Format, Added Outside System Section, Defined subsystems, | Emmanuel Girin |
| 2 | 06/09/2020 | Added output data | Leonardo Elias |
| 3 | 06/09/2020 | Finish Subsystem, Subsystem Diagram, Data Processing Steps, Context Diagram | Emmanuel Girin |
| 4 | 06/09/2020 | Added possible risk and risk mitigation | Leonardo Elias |
| 5 | 06/09/2020 | Updated and formatted the input data and output data tables. | Jacob Valentine |
| 6 | 06/09/2020 | Reviewed possible risk and mitigation  Added to possible enhancements | Leonardo Elias |
| 7 | 06/09/2020 | Added to risk and mitigation/ Possible enhancements | Marques Young |

Outside System

The Hotel Reservation Software will be a java application that runs on a client machine. The java application will have a GUI, which a user, usually a hotel staff member, uses to interact with the system. The java application will access a database that is located on the client machine to retrieve information. In future iterations of this software, there is a plan to incorporate a server hosted database. There will be the optional added user, presumably an IT tech, manager, or data analyst, which has special direct access to the database to change records bypassing the java application completely. In this analysis we will focus specifically on one of these outside systems that of the hotel employee, user, which primarily accesses the hotel reservation system to access reservations.

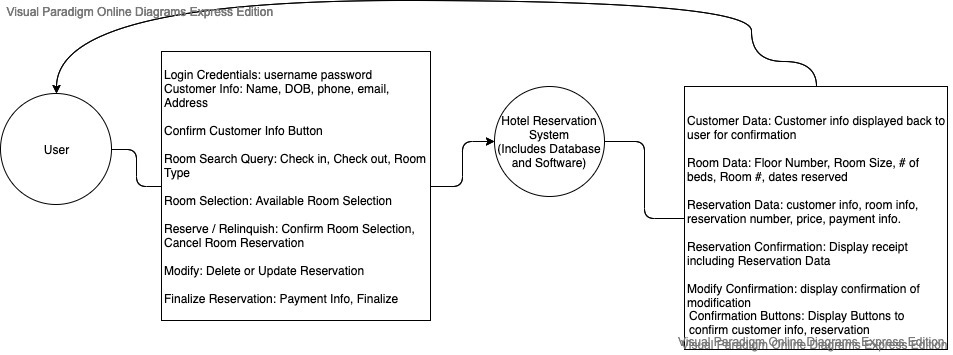
# Input Data

|  |  |  |
| --- | --- | --- |
| Data | Description | Source |
| Login credentials | A username and password associated with a system account. | User |
| Login Verification | A check from the central server saying whether given login credentials match an account on the central server, and what user level and account it corresponds to. | hosting SQL Database |
| Customer info | Personal information used to assign consumers to room reservations, including: first and last name, date of birth, phone #, address, and email. For different functions, only some or all of this information shall be required. | User |
| Room information | Information related to an individual room’s size, location, price, availability, and style, which is used to create new room entries and to modify existing ones. | User, hosting SQL Database |
| Room Search Query | Input information used to search rooms by location, availability, size, and price. | User |
| Room Selection | A selection of a specific room, which a reservation may be placed, updated, or deleted. | User |
| Reserve/Relinquish | The User puts in a request for the system to reserve/relinquish the room, using the provided information. | User |
| Reserve/Relinquish Verification | A check from the server to verify whether the reservation/relinquishment was successful. If it was not, information for why should be provided. | hosting SQL Database |
| Modify Reservation | A request to modify a room reservation. | User |
| Modify Verification | A check from the server to verify whether the modification was successful. If it was not, information for why should be provided. | hosting SQL Database |
| Modify Entry | A request to modify a room entry. | User |
| Delete | A request to delete a room entry. | User |

Output Data

|  |  |  |
| --- | --- | --- |
| Data | Description | Destination |
| Customer Data | A display showing an existing customer’s first and last name, date of birth, phone #, address, and email. | User, hosting SQL Database |
| Room Data | Displayed look up information for an individual room or rooms, including floor number, Room size, number of beds, Room number, dates reserved, and price. | User |
| Reservation Data | A display of the reservation status of given rooms. If the room is reserved, it should display both Customer information linked to the room (first and last name, date of birth, phone #, address, and email), as well as the information of the room itself (Floor number, room number, number of beds, room number and dates reserved).  In addition, the system should display Auto generated information such as a Reservation number, Price of the total reservation, and payment info. | User |
| Reservation Confirmation | Display confirmed reservations.  Email/Print reservation confirmation to customer. | User |
| Modify Confirmation | A confirmation that the reservation has been modified successfully, complete with the updated reservation data. | User |
| Login credentials | A username and password associated with a system account. | hosting SQL Database |

Context Diagram



# Data Processing Steps

At the main screen, a form is displayed where a user must type in a username and password. After entering the information, the user clicks a button to validate the data. The username and password are sent to the UserRecords database for validation. If a failure returns a try again message is displayed. Else progresses to the Customer screen.

The customer screen displays a choice of Creating a New Customer or Existing Customer. The user clicks on either button. If creating a new customer is selected the CreateCustomer screen is displayed. Else the customer Lookup screen is displayed.

* CreateCustomer screen displays an editable form for customer information (see above for the details). Once the user has filled in the information, user clicks a button, then the information is displayed in a locked form, user is asked to continue or edit. If user chooses edit form is redisplayed as unlocked for details to be changed. Else reservation screen displays.
* Customer lookup screen renders a form requesting name, phone number, or email of customer. Once user has entered information in the form, the information is sent to the customerRecords database for confirmation of existing user. If the customer does not exist a negative message will be displayed, and user will either try again or click a button to createNewCustomer. Else the customer information will return display, and the reservation screen displays.

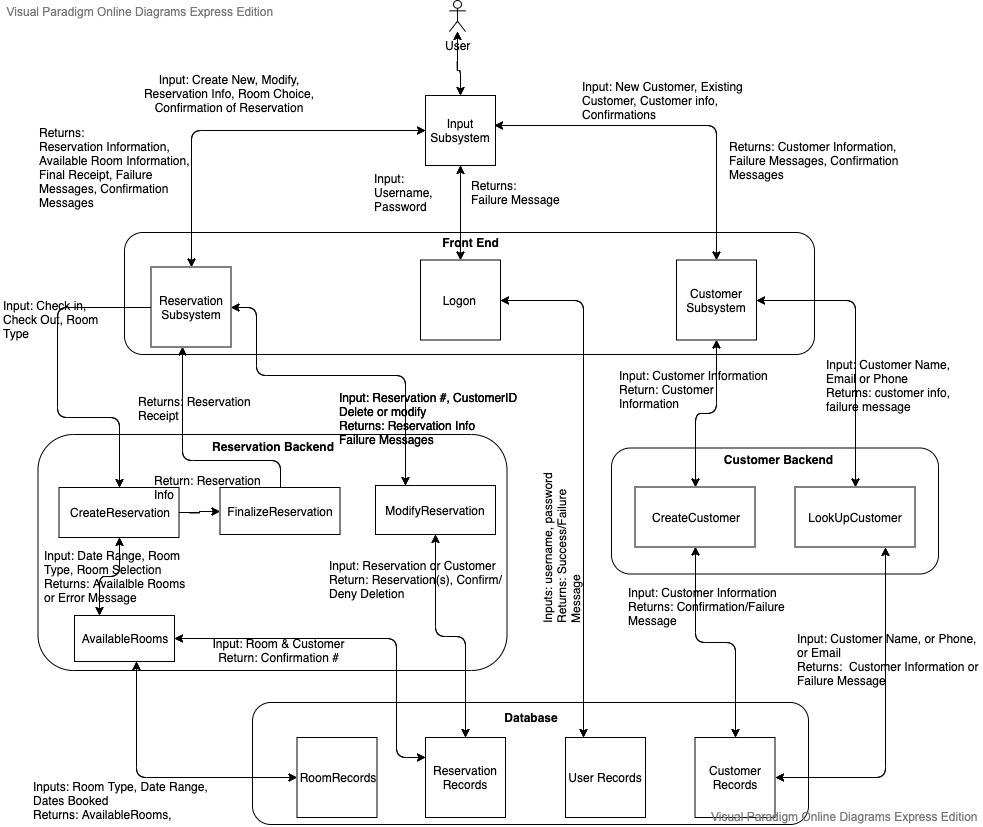
The reservation screen displays the choice of creating new reservation or modifying existing one. If user chooses creating new reservation the Create Reservation screen displays. Else the Modify Reservation screen displays.

* Create Reservation Screen displays a request for Check in Date, Check out Date, Preferred Room Type, and number of Rooms. Once User types in information and clicks confirm, the room data and dates are sent to the Available Rooms subsystem.  
  Available Rooms sends Date Data and preferred room type to RoomRecords. RoomRecords returns the rooms that are available. AvailableRooms displays available rooms to user and price. User selects the room needed. User is prompted to finalize reservation, which sends user to finalize reservation screen. Room information is sent to reservationRecords and roomRecords to update the booking of the room and the reservation data.
  + Modify Reservation Screen displays a form to request reservation confirmation # or perform a lookup of all reservations under customer
  + If user does not have reservation confirmation #, the customer ID info is sent to reservationRecords to return the data of the reservation(s) for that customer.
    - If there are no reservations, a failure message is displayed. User is returned to Create a Reservation screen.
    - Else a list of 1 or more reservations is displayed. User must select from the choice of reservations displayed.
  + If user does have a confirmation # data is sent directly to reservationRecords returning the data of the reservations for that customer. If there are none a failure message is displayed, and user can try again or return to create a Reservation screen.

If a reservation has been found from either of these options above, the user is displayed a choice to delete or modify the reservation. In either case the reservation is deleted from reservationRecords and roomRecords has the availability dates of the room updated. In the case of modifyReservation, the user is sent back to the Create Reservation Screen to make a new Reservation. In the case of delete Reservations, a confirmation is displayed to user that the reservation has been deleted, and the user is sent back to the main reservation screen.

The Finalize Reservation screen appears after a new reservation is created which displays room info, date info, customer info, and price to user in a form, which can be printed for a receipt. Afterwards user is sent back to main screen.

Subsystem Diagram



Subsystem Descriptions

The following subsystems pass along input and output data. Please refer to the input and output data sections for more specific detailing of the data.

1. Input – Receives input from user involving customer information, reservation information, credentials for logon, and sends output back to the user.
2. Logon – Displays username and login. Takes the input of username and password from user. Displays try again if password is wrong else accesses the Customer subsystem.
3. Customer – Displays existing or new customer. Receives user input. Links to CreateCustomer or LookupCustomer based on choice.
4. CreateCustomer - Displays a request for customer information through a form. Receives user input. Displays data to user for confirmation. If user input does not confirm gives chance to edit customer information form. Upon receiving confirm from user input sends data to Customer Records.
5. LookUpCustomer - Displays request for customer information from user, receives user input, and sends data to CustomerRecords for retrieval of customer data. Displays customer information in form and asks for confirmation of customer. Upon confirmation links to Reservation subsystem.
6. Reservation - Displays new reservation or modify existing reservation. Receives input from user. Links to CreateReservation or ModifyReservation subsystems based on choice
7. CreateReservation - Displays Request for Check In, Check Out, Preferred Room Type. Accepts user input for those values. Sends Data to Available Rooms
8. ModifyReservation - Displays Request for Reservation Info, Receives User Input for either first,last, & Dob, phone number, email, or reservation #. Sends data to ReservationRecords. Displays reservation data. Receive user input for whether to delete or change reservation. If delete send data to ReservationRecords and RoomRecords for update and link to Reservation. Else delete reservation by sending data to ReservationReocrds and RoomRecords, and link to CreateReservation.
9. AvailableRooms - Sends Date Data and Preferred Room Type to RoomRecords. Displays Available Rooms and Price. Accepts user input to select a room & confirms choice. Sends room choice to RoomRecords and ReservationRecords. Send Reservation Data to FinalizeReservation subsystem
10. FinalizeReservation - Displays Reservation Info. User can print receipt information and navigate back to main page.
11. CustomerRecords - Accepts new customer request with customer information data. Creates a new entry in the records. Sends Customer information data back to the calling subsystem. Accepts customer info for look up purposes. Sends back existing customer profile or default failure message.
12. ReservationRecords - Accepts CreateReservation requests with reservation data. Creates new reservation entry. Returns reservation data to subsystem. Accepts ReservationLookup requests with reservation data. Returns existing reservation data back to calling subsystem.
13. RoomRecords - Accepts look up data for room type, checkin, checkout for lookup. Returns available rooms within dates and type. Accepts room # and checkin and check out. Flags the room as booked. Returns confirmation that room is booked.
14. UserRecords - Username and Password File table, updated separately by IT staff. Receives request to confirm correct username and password. Returns confirmation, yes or no of correct username and password.

# Subsystem-Requirement Implementation

|  |  |  |
| --- | --- | --- |
| **#** | **Requirement** | **Subsystem** |
| 1 | Software shall store personal consumer information (first and last name, date of birth, phone #, address, email) | Input, CreateCustomer, CustomerRecords |
| 2 | Software shall protect access to consumer information through username and password access | Input, Logon UserRecords |
| 3 | Software shall use SQL database to store consumer information | Create Customer, CustomerRecords |
| 4 | SQL Database shall contain three tiers of users | UserRecords |
| 5 | SQL Database shall have a master user that has super access to pass off to IT Team | UserRecords |
| 6 | SQL Database shall have an mid-level user that can view and edit reservations / customer data | UserRecords |
| 7 | SQL Database shall have a base-level user that can only view reservations (or maybe mid-level can delete reservation like manager access, and employee base level can only view and make reservations) | UserRecords |
| 8 | Software shall allow a hotel staff to make a reservation for a guest | Create Reservation, Database |
| 9 | Software shall store reservations for rooms on specific dates | Create Reservation, ReservationRecords |
| 10 | Software shall display available rooms on dates | AvailableRooms |
| 11 | Software shall have a basic UI with Menu | Input |
| 12 | Menu shall display option for Finding Existing Reservation, Making new Reservation, deleting reservation and updating reservation. | Input, Reservation, CreateReservation, ModifyReservation |
| 13 | Software shall require customer information to be captured when making a new reservation | Customer, Create Customer, LookUpCustomer |
| 14 | Software shall ask staff it the customer already exists, and perform a look up by name, phone number, or email address | LookUpCustomer |
| 15 | Software shall give Staff option to capture new customer Data if customer does not exist in the database (Register a new customer) | Create Customer |
| 16 | Existing reservations shall be performed by a customer look up | ReservationLookup, LookUpCustomer |
| 17 | Software shall give staff the ability to change a reservation to a new date, room, and location (update reservation) | ModifyReservation |
| 18 | Software shall give staff ability to delete reservation | ModifyReservation |
| 19 | To Update or Create new Reservations: Staff shall enter date, which shall return a list of available room and descriptions | AvailableRooms, CreateReservation, ModifyReservations, FinalizeReservation. |
| 20 | Rooms shall be organized by size (single queen bed, two double beds, single king, suite, and master suite) | RoomRecords |
| 21 | UI shall be simple to use Graphics UI with text boxes to enter in data and buttons to navigate and perform lookup functions | Input |
| 22 | Software shall ask for Credential Logon at beginning of each use to ensure security | Logon |

# Possible Enhancements

There are several enhancements that can be added to make for a better reservation process. One of the first enhancements to be made is to give the IT department database privileges in the case that an employee is unable to login with their credentials.

While best practice is least privilege for most employees, managers should be given sole permissions to approve delete reservation requests. This allows the manager to potentially save a sale and/or prevents an employee from unintentionally ruining a customer’s reservation. Managers will also be the only individuals allowed to access customers’ sensitive information.

Existing customers’ information should be left editable in the case that their personal identifiable information was entered incorrectly; this information includes name, phone number, email address, home address, and payment methods.

While the program is built for a small hotel/motel, it does not account for the potential for multiple workstations that have access to booking information. Since this is the case, it is vitally important to check concurrency issues within the database so that rooms are not able to be double booked. This would be something as simple as blacking out rooms that are already booked for certain days.

# Possible Risks and Risk Mitigation

One of the possible risks associated with almost any database or web application is a SQL injection. A SQL injection happens when a third party can use SQL commands to interfere with back-end databases in ways that they shouldn’t be allowed. It generally occurs when websites directly incorporate user-inputted data into an SQL query, the query is then run against a database. There are a few ways to mitigate this type of attack. One of the key ways to mitigate this risk is to use appropriate privileges; don’t connect your database to an account with admin-level privileges unless necessary. You should also get rid of any unnecessary database functionality as this increase “surface size” and makes the database more vulnerable to SQL injections.

Another possible risk is what is known as a brute force attack. This is essentially someone trying to use every password or passphrase until they can gain access to material, they shouldn’t have access to. One of the best practices for preventing a brute force attack is to lockout an account after a set number of attempts, preferably no more than three. A secondary way to prevent a brute force attack is to delay the response time between password attempts. This allows administrators more time to detect potential intrusion.

Multi-factor authentication essentially adds layers to security. Instead of just a username and password, an email address can be used to verify that you are who you say you are; MFA compensates for the weakness of a password or passphrase. MFA also helps reach compliance standards that are established. Having MFA allows for more advanced logins like single sign-on; single sign-on works by validating the user through MFA, once the is authenticated they have access to all the covered apps through the SSO software.

Authentication requirements:

* The password's length must be greater than 8 characters and less than 64 characters.
* Allow all the characters including Unicode and whitespace.
* Implement secure password recovery which allows users to gain access to their account in case they forget their password.

# Project Design

## Revision Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Description** | **Contributor** |
| 0 | 06/14/2020 | Creating Document, Cover Page, Revision Table, Event Trace Section, and Class Designs section | Jacob Valentine |
| 1 | 06/15/2020 | Added Class Design Details | Emmanuel Girin |
| 2 | 06/16/2020 | Finalizing Class Designs, including GUI design | Emmanuel Girin |
| 3 | 06/16/2020 | Startup/ Shutdown | Leonardo Elias |
| 4 | 06/16/2020 | Added Normal Scenarios Pre/Post Conditions, Descriptions, and steps | Marques Young, Jacob Valentine |
| 5 | 06/16/2020 | Added Event Trace Diagrams | Leonardo Elias |
| 6. | 06/16/2020 | Risks/Mitigation | Marques Young |

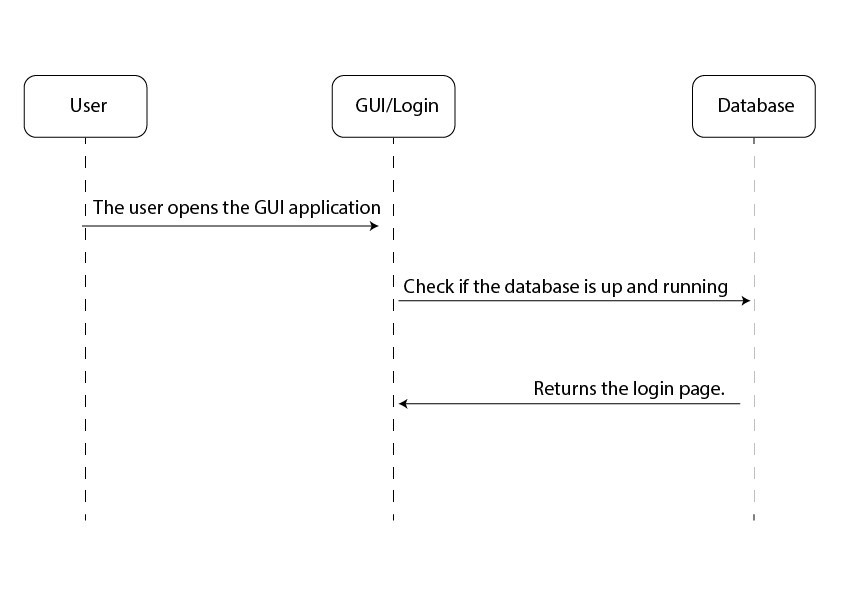
# Event Trace Diagrams

## Normal Scenarios

### Scenario 1 – Startup

Description:

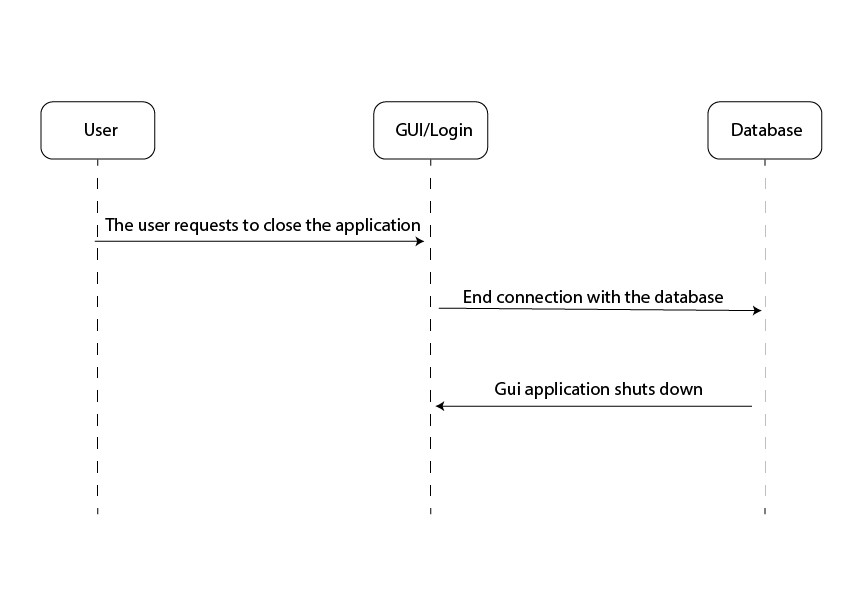
1. The user opens the application.
2. GUI prompts for login.
3. Check if the database is up and running.
4. It returns the login page.



Scenario 2 – Shutdown

Description:

1. The user requests to close the application.
2. End connection with the database.
3. Gui application shuts down.



### Scenario 3 – Create new Customer

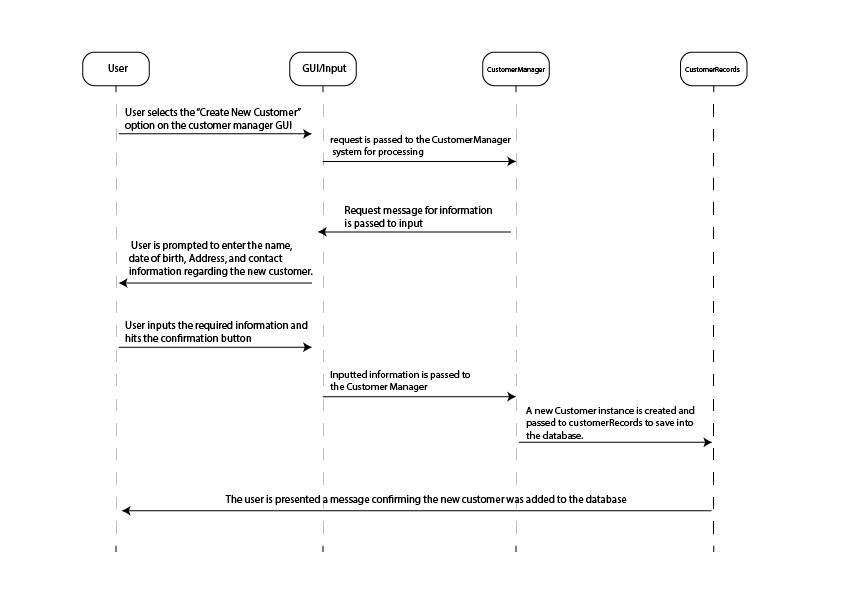
Pre-condition: the user selects the “Create new Customer” option from the Customer Manager

Post-Condition: The customer entry is added to the customerRecords and the user is shown a confirmation message.

Description: After logging in, the user can create a new Customer entry to be saved into the database by selecting the corresponding option, and inputting the required customer information (Name, date of birth, contact information, etc.).

Steps:

1. [User -> input/gui] User selects the “Create New Customer” option on the customer manager GUI
2. [Input/Gui -> CustomerManager] request is passed to the CustomerManager system for processing
3. [CustomerManager -> Input/Gui] Request message for information is passed to input
4. [Input/gui -> User] User is prompted to enter the name, date of birth, Address, and contact information regarding the new customer.
5. [User -> Input/GUI] User inputs the required information and hits the confirmation button
6. [Input/GUI -> CustomerManager] Inputted information is passed to the Customer Manager
7. [CustomerManager -> CustomerRecords] A new Customer instance is created and passed to customerRecords to save into the database.
8. [CustomerRecords -> CustomerManager] a Confirmation is passed
9. [CustomerManager -> Input/gui] a confirmation is passed
10. [Input/GUI -> User] The user is presented a message confirming the new customer was added to the database.



### Scenario 4 – Select an existing Customer

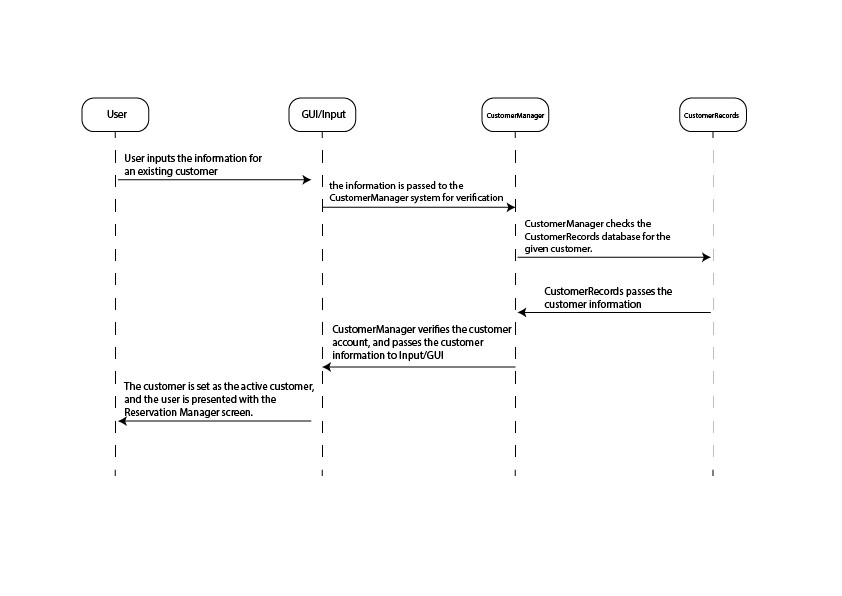
Pre-Condition: The user inputs the account information linked to an existing customer

Post-Condition: The selected customer is selected as the active customer, and the reservation manager is displayed.

Description: The user selects a customer from the database of customers and manages that customers' existing and future reservations from there.

Steps:

1. [User -> input/gui] User inputs the information for an existing customer
2. [Input/Gui -> CustomerManager] the information is passed to the CustomerManager system for verification
3. [CustomerManger -> CustomerRecords] CustomerManager checks the CustomerRecords database for the given customer.
4. [CustomerRecords -> CustomerManager] CustomerRecords passes the customer information
5. [CustomerManager -> Input/Gui] CustomerManager verifies the customer account, and passes the customer information to Input/GUI
6. [Input/gui -> User] The customer is set as the active customer, and the user is presented with the Reservation Manager screen.



### Scenario 5 – Update Customer Entry

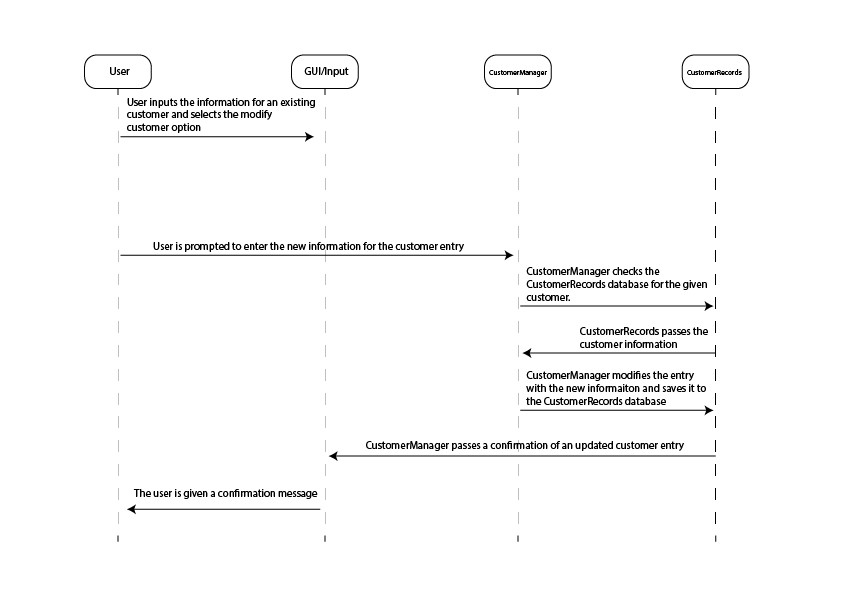
Pre-Condition: The user inputs the information linked to an existing customer entry, and selects the “Modify Customer Account” option.

Post-Condition: The Customer entry of the CustomerRecords database is updated and the user is given a confirmation message.

Description: The user can modify an existing customer entry with updated information, should they need to.

Steps:

1. [User -> input/gui] User inputs the information for an existing customer and selects the modify customer option
2. [Input/gui -> User] User is prompted to enter the new information for the customer entry
3. [User -> Input/gui] User inputs the new information
4. [Input/Gui -> CustomerManager] the information is passed to the CustomerManager system
5. [CustomerManger -> CustomerRecords] CustomerManager checks the CustomerRecords database for the given customer.
6. [CustomerRecords -> CustomerManager] CustomerRecords passes the customer information
7. [CustomerManger -> CustomerRecords] CustomerManager modifies the entry with the new informaiton and saves it to the CustomerRecords database
8. [CustomerRecords -> CustomerManager] passes a confirmation
9. [CustomerManager -> Input/Gui] CustomerManager passes a confirmation of an updated customer entry
10. [Input/gui -> User] The user is given a confirmation message.



### Scenario 6 – Lookup reservation by customer and checkin date

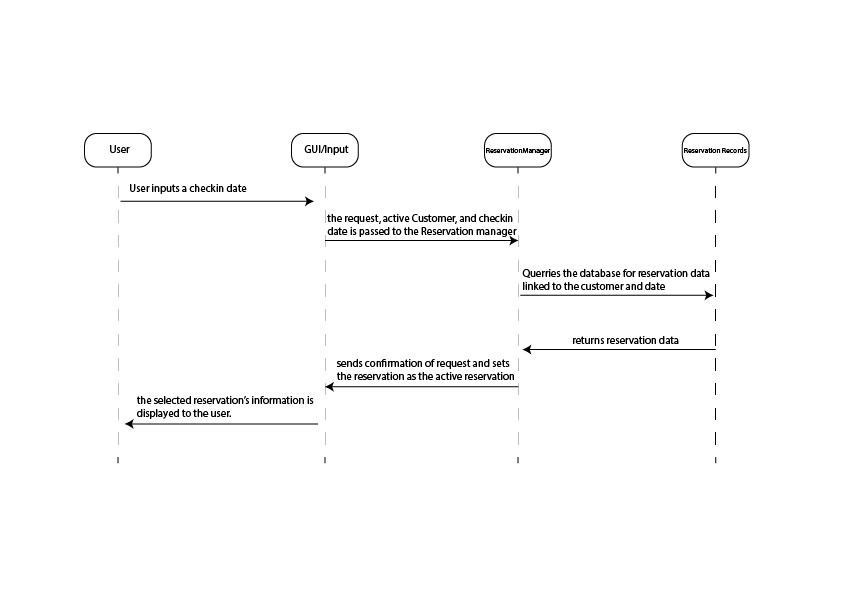
Pre-Condition: The user inputs the checkin date for a reservation linked to the active customer

Post-Condition: The information for a Reservation linked to the active customer is displayed and selected.

Description: The user is able to look up a room reservation by customer for ease of use.

Steps:

1. [User -> Input/GUI] User inputs a checkin date
2. [Input/GUI -> ReservationManager] the request, active Customer, and checkin date is passed to the Reservation manager.
3. [ReservationManager -> Reservation Records] Querries the database for reservation data linked to the customer and date
4. [ReservationRecords -> ReservationManager] returns reservation data
5. [Reservation Manager -> Input/GUI] sends confirmation of request and sets the reservation as the active reservation
6. [Input/GUI -> User] the selected reservation’s information is displayed to the user.



### Scenario 7 – Lookup reservation by ID

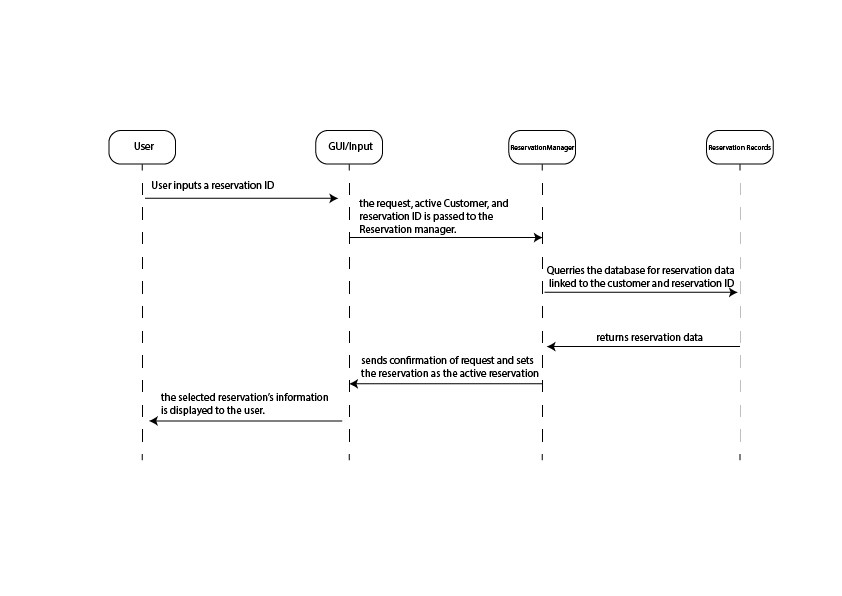
Pre-Condition: The user inputs an ID to search for a reservation.

Post-Condition: The reservation is displayed and selected based on the entered ID.

Description: This allows a user to look up a reservation by entering an ID instead of using a customer’s name.

Steps:

1. [User -> Input/GUI] User inputs a reservation ID
2. [Input/GUI -> ReservationManager] the request, active Customer, and reservation ID is passed to the Reservation manager.
3. [ReservationManager -> Reservation Records] Querries the database for reservation data linked to the customer and reservation ID
4. [ReservationRecords -> ReservationManager] returns reservation data
5. [Reservation Manager -> Input/GUI] sends confirmation of request and sets the reservation as the active reservation
6. [Input/GUI -> User] the selected reservation’s information is displayed to the user.



Scenario 8 – Delete Reservation

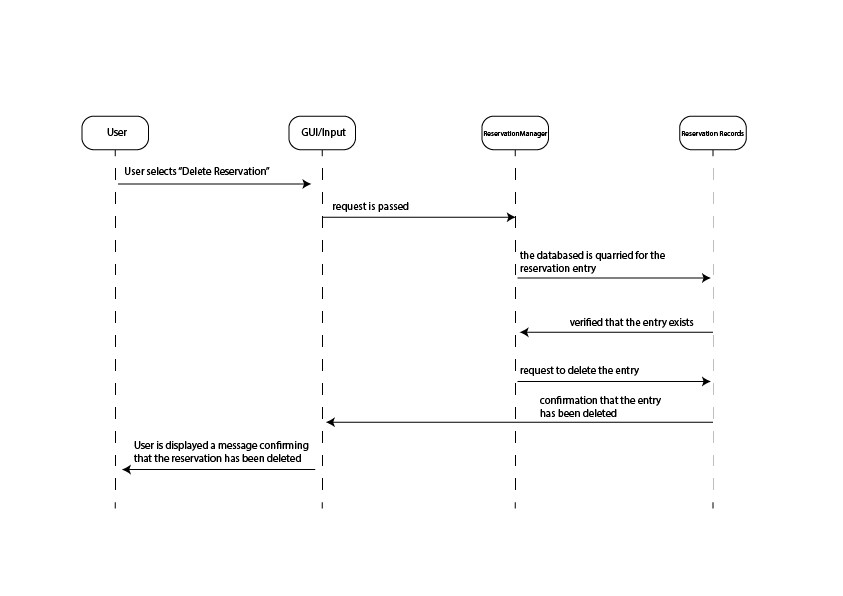
Pre-Condition: The user selects the “Delete Reservation” option with an active reservation selected.

Post-Condition: the user is presented with confirmation that the reservation has been deleted.

Description: This allows a user (manager) to delete an existing reservation if the need arises.

Steps:

1. [User -> Input/GUI] User selects “Delete Reservation”
2. [Input/GUI -> ReservationManager] request is passed
3. [ReservationManager -> ReservationRecords] the databased is quarried for the reservation entry
4. [ReservationRecords -> ReservationManager] verified that the entry exists
5. [ReservationManager -> ReservationRecords] request to delete the entry
6. [ReservationRecords -> ReservationManager] confirmation that the entry has been deleted
7. [ReservationManager -> Input/GUI] Confirmation is passed
8. [Input/GUI -> User] User is displayed a message confirming that the reservation has been deleted.



### Scenario 9 – Lookup available Rooms

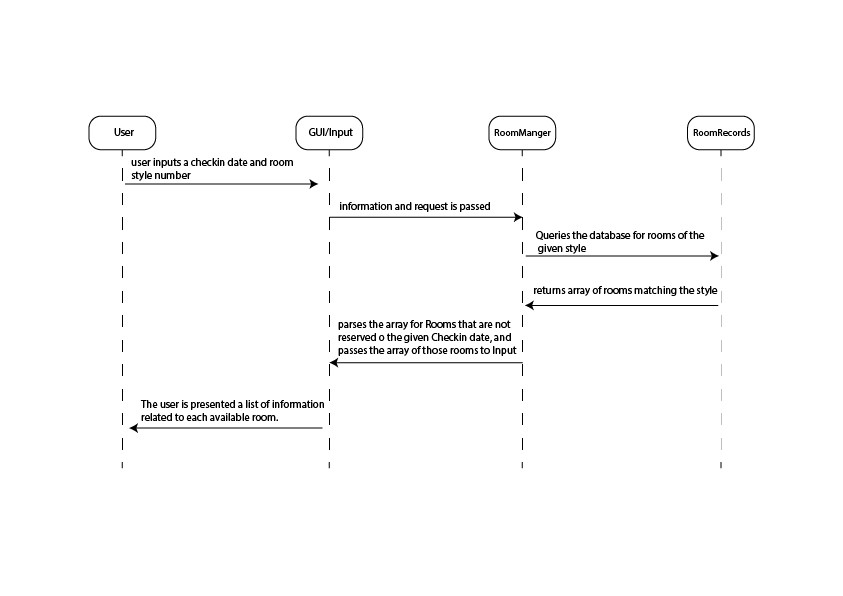
Pre-Condition: The user inputs a checkin date, room style (number of beds), has an active customer selected, and selects “Display available rooms”

Post-Condition: The user is presented a list of information for all rooms not reserved

Description: The user can easily see what rooms of a given style are available for reservation on a given day.

Steps:

1. [User -> Input/GUI] user inputs a checkin date and room style number
2. [Input/GUI -> RoomManger] information and request is passed
3. [RoomManager -> RoomRecords] Queries the database for rooms of the given style
4. [RoomRecords -> RoomManager] returns array of rooms matching the style
5. [RoomManager -> Input/GUI] parses the array for Rooms that are not reserved o the given Checkin date, and passes the array of those rooms to Input
6. [Input/GUI -> User] The user is presented a list of information related to each available room.



### Scenario 10 – Reserve Room

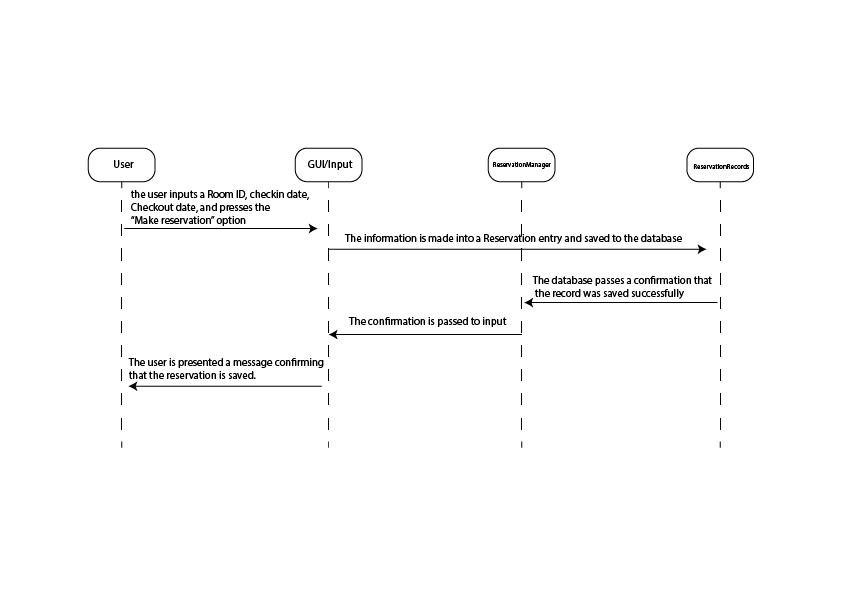
Pre-Condition: User inputs the Room ID for an available room, checkin date, and checkout date, with an active customer selected.

Post-Condition: The new reservation is made and the user is shown a confirmation message.

Description: The user can reserve available rooms for customers within given checkin and checkout days.

Steps:

1. [User -> Inputs/GUI] the user inputs a Room ID, checkin date, Checkout date, and presses the “Make reservation” option
2. [Input/GUI -> ReservationManager] information is passed
3. [ReservationManager -> ReservationRecords] The information is made into a Reservation entry and saved to the database
4. [ReservationRecords -> ReservationManager] The database passes a confirmation that the record was saved successfully
5. [ReservationManager -> Input/GUI] The confirmation is passed to input
6. [Input/GUI -> User] The user is presented a message confirming that the reservation is saved.



## Error Handling Scenarios

### Scenario 1 – Login Failure

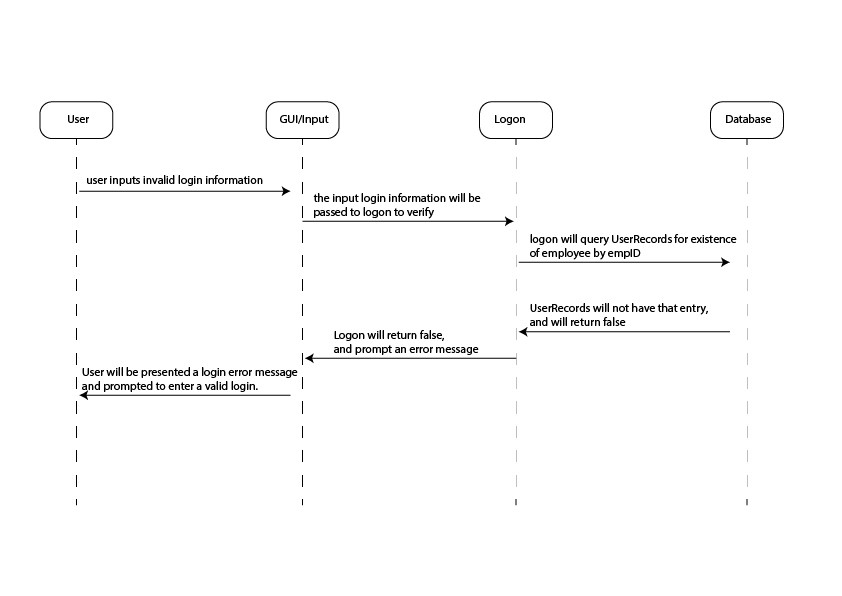
Pre-Condition: User inputs invalid login information

Post-Condition: User will be presented with an “Invalid Login”

Description: The user provides an invalid username or password when prompted for a login. If the username/Password combination is not found, an error message will be displayed to the user telling them that either the username or password is incorrect, and will be prompted to enter a valid login.

Steps:

1. [User -> input/gui] user inputs invalid login information
2. [input/gui -> Logon] the input login information will be passed to logon to verify
3. [logon -> UserRecords] logon will query UserRecords for existence of employee by empID
4. [UserRecords -> Logon] UserRecords will not have that entry, and will return false
5. [logon -> input/gui] Logon will return false, and prompt an error message
6. [input/gui -> user] User will be presented a login error message and prompted to enter a valid login.



### Scenario 2 – Database disconnected

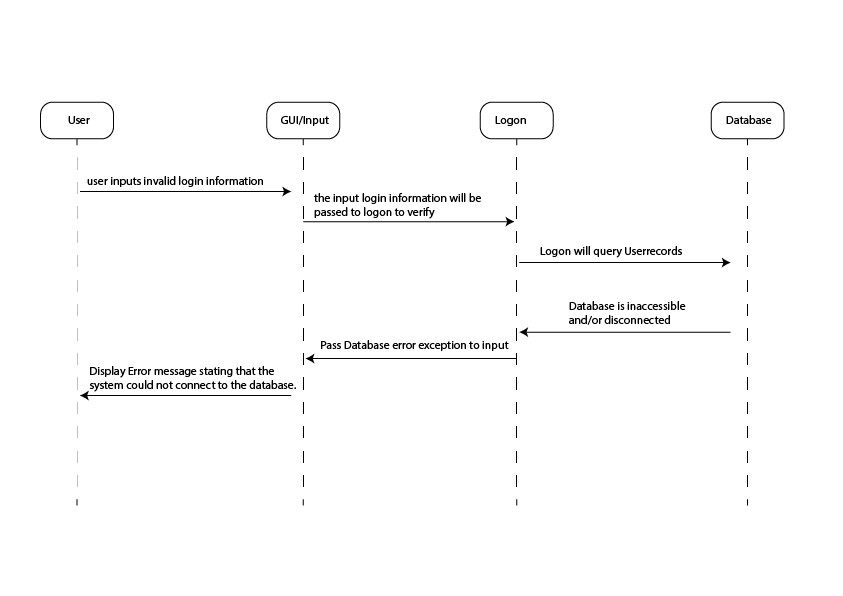
Pre-Condition: The User attempts to login, but the User lookup database is inaccessible

Post-Condition: The user will be given an error message stating that the application could not access the database, and to try again later.

Description:

Steps:

1. [User -> Input/gui] user inputs login information
2. [input/gui -> Logon] the input login information will be passed to logon to verify
3. [Logon -> UserRecords] Logon will query Userrecords
4. [UserRecords -/> Logon] Database is inaccessible and/or disconnected
5. [Logon -> input/gui] Pass Database error exception to input
6. [input/gui -> user] Display Error message stating that the system could not connect to the database.



## Class Designs

### Input / GUI Subsystem

This class writes the whole Graphical In / Out interface for the USER. As this class may develop into further classes to avoid becoming too large, we have loosely defined the graphics pseudocode to describe the flow.

This GUI scenario assumes a typical flow through the program with Main Features highlighted

1. load LoginGUI: Login FORM asking for Employee ID and Password
2. user submits form through button: data is passed to validateCredentials
3. Login is authenticated -> User is brought to CustomerSelect Page
4. User Selects Existing Customer or New Customer
5. User Selects New Customer -> New Customer Form is loaded
6. User enters customer information selects button to submit form data
7. Data is passed to CustomerManager, which creates a new customer in database
8. ReservationSelection is Displayed – Asking user to choose between existing or new reservation
9. User chooses New Reservation Button
10. Load ReservationCreate Form – with form for checkin, checkout, room type
11. Create RoomsManager Object to find AvailableRooms and pass info back
12. Display AvailableRooms to User for Selection
13. User Selects Available Rooms, which are passed to Reservation Manager to create New Reservation
14. The info for the reservation is displayed and if user hits finalize, the reservation will get created in the database and receipt info will be displayed.

The GUI for this program will have a main frame created in the first display method of the class below. The main frame will have an exit button. IT will leave the entire center for panels. Each display call will update the panel in the center, simulating a new screen. **The logic of this GUI is represented by these main methods, however there are many button implementations that navigate between these screens. These will be hard coded in to navigate between these screens.**

### HotelReservationGUI

|  |
| --- |
| **class HotelReservationGUI** {  Frame frame;  ReservationManager reservMgr;  CustomerManager custMgr;  RoomManager roomMgr;  Login logon;  int LogonAttempts = 0;    //methods  static void main(int param) {  HotelReservationGUI a = new HotelReservationGUI;  a.displayLogin();  }  void displayLogin(){  create Frame with exit button  create panel with form  //form with empID, password label and textboxes and a button to login  render loginGUI;  If (empID has characters & password has characters)  enable Login Button;  Login Button onButtonClick  {  else If ( logon.ValidateCredentials(empID, password) )  displayCustomerManager();  else {  //error Message  display logon.getMessage();  //reset display  displayLogin();  logOnAttempts++;  //check for too many login attempts  If (logOnAttempts >= 5) {  Display Error Message;  Exit System;  }  }  }  } //end method  void displayCustomerManager{  render selectCustomerGUI  button newCustomer  button existingCustomer  custMgr = new CustomerManager();  if User chooses NewCustomer Button  displayNewCustomerForm();  else if User chooses existingCustomer Button  displayExistingCustomerForm();  }  void displayNewCustomerForm{  render newCustomerForm  Form Displays textboxes for firstName, lastName, street, city, state, country, phone,  email;  Form Data is input into String Array customerData;  custMgr.createNewCustomer(customerData);  displayReservationManager();  }//end method  void displayExistingCustomerForm{  render existingCustomerForm  Form Displays textboxes for firstName, lastName, phone, email;  button LookUpByName, LookUpByEmail, LookUpByPhone;  boolean continue = false;  //decide which lookUpToPerform  LookUpByName Clicked {  if (firstName & lastName are filled)  continue = custMgr.lookUpCustomerByName(first, last);  if(continue)  displayReservationManager();  else  display “Failed to find customer. Exit or Try Again”  }  LookUpByPhone clicked {  if (phone is filled)  continue = custMgr.lookUpCustomerByPhone(phone);  if(continue)  displayReservationManager();  else  display “Failed to find customer. Exit or Try Again”  }  LookUpByEmail clicked {  if(email is filled)  continue = custMgr.lookUpCustomerByEmail(email);  if(continue)  displayReservationManager();  else  display “Failed to find customer. Exit or Try Again”  }  }//end method  void displayReservationManager {  renderReservationManager  button NewReservation, Existing Reservation;  display buttons  if user clicks NewReservation  displayNewReservation();  if user clicks ExistingReservation  displayExistingReservation();  }//end method  void displayExistingReservation{  render ExistingReservation  Form asking for CheckInDate, reservation number;  button LookUpbyName, LookUpByNumber;  if user clicks LookUpByName {  try {  reserveMgr.LookUpByName(custMGR.getCustID(), checkInDate);  displayReservationData();  }  catch Exception Display Message to User  }  if user clicks LookUpByNumber {  try {  reserveMgr.LookUpByReservation(reservationNumber);  displayReservationData();  }  catch Exception Display Message to User  }  }//end method  void displayReservationData  {  button ModifyReservation, DeleteReservation  displays reserveMGR.getCurrentReservation.toString() in a textbox  if ModifyReservation button is clicked {  //deletes existing reservation  reserveMgr.deleteReservation();  //navigate to create a New Reservation, replacing old one  displayNewReservation();  }  if DeleteReservation button is clicked {  reserveMgr.deleteReservation();  //navigate back to Customer Manager Screen to start over  displayCustomerManager();  }  }//end method  void displayNewReservation {  render NewReservation  Form Asking for: checkIn, checkOut, # of Guests,Dropdown Preferred Room Type, # of  rooms;  roomsMgr = new RoomManager (checkIn, checkOut, Prefered Type, numRooms)  try {  roomsMgr.SearchForAvailableRooms();  //if no exceptions thrown then rooms have been found  reserveMGR.createReservation(checkIn, checkOut, int numGuests,  roomsMgr.getSelectedRooms());  displayFinalizeReservation();  }  catch Exceptions and display appropriate Message, redisplay Form  }//end method  void displayFinalizeReservation(); {  button finalizeReservation, cancel;  textbox reservationDetails  Display reserveMGR.getCurrentReservation().toString() in reservationDetails  if finalizeReservation is clicked {  reserveMGR.finalizeReservation  //shows the newly created Reservation ID.  // Allows user to print reservation details  display reserveMGR.getCurrentReservation().toString() in textbox  display button exit  if exit is clicked  SystemExit();  }  if cancel is clicked {  //return to CustomerManager page  displayCustomerManager();  }  }//end method  }//end class |

Logon Subsystem

|  |
| --- |
| **static class Login** {  //data private String message;  //methods  String getMessage() {  return message;  }//end method  boolean ValidateCredentials(String empID, String password)) {  connect to database  query UserRecords for existence of employee by empID  If (database returns true) {  compare (password) to UserRecord employee’s password  If (database returns true) {  return true;  }//end inner if Else {  message = “Error Incorrect Password”;  return false;  }  } //end outer if  Else {  message = “Incorect EmployeeID”;  return false;  }  } //end method  }//end class |

Customer Subsystem

This subsystem combines previous subsystems outlined for customers into 2 classes. The customer manager class accesses the database to look up an existing customer, modify existing customer data, or create a new customer. This customer manager class combines our 2 previous subsystems of creating a new customer and looking up an existing one. The customer object is where the data is stored either from input or from the database depending on the method needed.

|  |
| --- |
| **class CustomerManager** {  //data  private Customer currentCustomer;  //methods  void createCustomer (StringArray CustomerData) {  currentCustomer = new Customer(CustomerData);  connect to Database customerRecords;  //Adds all of the data points into Database Table CustomerRecords  Create New Row in customerRecords(  currentCustomer.getFirstName(),  currentCustomer.getlastName(),  currentCustomer.getDob(),  currentCustomer.getstreet(),  currentCustomer.getCity(),  currentCustomer.getState(),  currentCustomer.getCountry(),  currentCustomer.getPhone(),  currentCustomer.getEmail()  );  //Retrieves the unique key from the database  currentCustomer.setCustomerID(retrieve customerID from customerRecords);  }//end method  boolean lookUpCustomerByName(String firstName, lastName) {  connect to Database customerRecords  querry customerRecords Table for match of firstName, lastName  if (matchFound){  retrieve customerData from Database  StringArray customerData = new StringArray size 10);  for each element in customerData {  add corresponding column info  }  currentCustomer = new Customer(customerData);  return true;  }  else  return false; //lets calling method know that lookUpFailed  } //end method  boolean lookUpCustomerByEmail(String email) {  connect to Database customerRecords  querry customerRecords Table for match of email  if (matchFound){  retrieve customerData from Database  StringArray customerData = new StringArray size 10);  for each element in customerData {  add corresponding column info  }  currentCustomer = new Customer(customerData);  return true;  }  else  return false; //look up Failed  } //end method  boolean lookUpCustomerByPhone(String phoneNumber) {  connect to Database customerRecords  querry customerRecords Table for match of phoneNumber  if (matchFound){  retrieve customerData from Database  StringArray customerData = new StringArray size 9);  for each element in customerData {  add corresponding column info  }  currentCustomer = new Customer(customerData);  return true;  }  else  return false; // look up Failed  } //end method  void modifyCustomer(StringArray CustomerData){  connect to customerRecords table;  query by CurrentCustomer.getCustomerID();  for each element in CustomerData {  Modify Column in current Customer Record with CustomerData[element];  }  }//end method  }//end class |

|  |
| --- |
| **class Customer** {  Private String CustomerID, firstName, lastName;  Private String dateOfBirth; // YYYY:MM:DD format  Private String street, city, state, country;  Private String phone, email //phone # will be formatted and validated possibly  //Constructor Takes a String Array for each point of data.  Public Customer (StringArray CustomerData) {  customerID = customerData[0]  FirstName = CustomerData[1];  LastName = CustomerData[2]; dateOfBirth = CustomerData[3];  Street1 = CustomerData[4];  City = CustomerData[5];  State = CustomerData[6];  Country = CustomerData[7];  phone = CustomerData[8];  email = CustomerData[9];  }  public toString() {  Display class variables;  }  //Methods to set Data  Void setCustomerID(String str) {  customerID = str;  }  Void setFirstName (String str) {  firstName = str;  }  Void setLastName (String str) {  lastName = str;  }  Void setDateOfBirth (Date dob) {  DateOfBirth = dob;  }  Void setStreet(String str) {  street1 = str;  }  Void setCity(string str) {  city = str;  }  Void setState(String str) {  state = str;  }  Void setCountry(String str) {  country = str;  }  Void setEmail(String str) {  email = str;  }  Void setPhone(String str) {  phone = str;  }  //Methods to get Data  String getCustomerID(){  return customerID;  }  String getFirstName () {  return firstName;  }  String getLastName() {  return lastName;  }  String getDob() {  Return dateOfBirth;  }  String getStreet() {  return street1;  }  String getCity () {  return city;  }  String getState () {  return state;  }  String getCountry() {  return country;  }  String getPhone() {  return phone;  }  String getEmail () {  return email;  }  } //end Class |

### Reservation Subsystem

The reservation subsystem combines 4 of our previous subsystems into 2 classes. The reservation class holds the data for all the reservations. The reservationManager class interfaces with the input and the database to create a reservation object and retrieve data from the database or upload data to the database.

|  |
| --- |
| **class Reservation** {  //data  private int ReservationID, customerID, numOfGuests;  private double totalCost;  private Int Array roomIDs;  private Date checkIn, checkOut;  private final taxRate = .10; //taxRate is set in this program  //methods  //constructor for reservations existing in database  public Reservation(int reservation, int cust, Int Array rooms, int cost, int guests,  date in, Date out) {  reservationID = reservation;  customerID = cust;  numOfGuests = guest;  roomIDs = rooms;  totalCost= cost \* (1 + taxrate);  checkIn = in;  checkOut = out;  }//end constructor  public toString() {  Display class variables;  }  //get methods for all data  int getReservationID() {  return ReservationID  }  int getCustomerID() {  return customerID  }  int getNumOfGuests() {  return numOfGuests;  }  intArray getRoomIDS() {  return roomIDs;  }  Date getCheckIn() {  return checkIn;  }  Date getCheckOut() {  return checkOut  }  } //end class |

|  |
| --- |
| **class ReservationManager {**  //data  Reservation currentReservation;  //methods  void createReservation(Date checkIn, Date checkOut, int numGuests, int array roomIDs) {  //find out total cost of 1 or more rooms per night  double totalPrice = 0;  int totalNights = checkOut – checkIn;  for each roomsSelected {  //retrieve price of selected rooms  totalPrice += roomsSelected[element].getPrice();  }  //Multiply total cost for the rooms per night by the amount of nights  totalPrice = totalPrice \* totalNights;  currentReservation = new Reservation(-1, int custID, Int Array roomIDs, double totalPrice, int  guests, date in, Date out);  }  void deleteReservation() throws databaseLookUpError {  query Database reservationRecords by currentReservation.get(reservationID);  if (recordFound)  delete record from database;  else  throw new databaseLookUpError();  }//end method  void lookUpReservationByCustomer(int customerID, checkInDate)  throws databaseLookUpError{  //retrieve reservation from Database make a reservation object  query Database ReservationRecords for record by customerID and checkInDate  if (recordFound) {  CurrentReservation = new Reservation (retrieve ReservationID, retrieve customerID,  retrieve roomIDs, retrieve checkIn, retrieve checkOut, retrieve Price, retrieve numGuests);  }  else  throw new databaseLookUpError();  }//end method  Reservation lookUpReservationByID(int reservationID) throws databaseLookUpError{  //retrieve reservation from Database make a reservation object  query Database ReservationRecords by reservationID  if(recordFound) {  CurrentReservation = new Reservation (retrieve ReservationID, retrieve customerID,  retrieve roomIDs, retrieve checkIn, retrieve checkOut, retrieve Price, retrieve numGuests);  }  else  throw new databaseLookUpError();  }//end method  void finalizeReservation () {  //Actually input reservation into database  connect to ReservationRecords  make new Row with (  currentReservations.getCustomerID(),  currentReservations.getRoomIDs(),  currentReservations.getCheckIn(),  currentReservations.getCheckOut(),  currentReservations.getPrice(),  currentReservations.getNumGuests()  )  //retrieve unique ReservationID  currentReservations.setReservationID = RoomRecords.retrieve current row key;  }//end method  void getCurrentReservation {  return currentReservation:  }//end method  }//end class |

### AvailableRooms Subsystem

The AvailableRooms subsystem is composed of 2 classes. A room class, which holds the data for a room object, as well as methods to get / set such data. Lastly it contains a roomManager class, which interfaces with the database, reservation manager, and input data to populate a list of available rooms for a specific date.

|  |
| --- |
| **class RoomManager** {  //data  private RoomArrayList availableRooms;  private int totalRooms;  Date checkIn, checkOut; //Possibly turn into string type later for compatibility with SQL  int preferredType; //Type of Room (0 = any, 1 = single bed, 2 = 2 beds, 3 = suite...)  int roomsNeeded; //# of rooms needed by customer  public RoomManager (Date in, Date out, int roomType, int quantity) {  checkIn = in;  checkOut = out;  preferredType = roomType;  roomsNeeded= quantity;  availableRooms = new RoomsArrayList();  totalRooms = querry Database roomRecords for total# of rows;  }//end constructor  //methods  void searchAvailableRooms () throws ExceedsMaxCapacityError(), unavailableReservationsError(); {  if (totalRooms < roomsNeeded){  throw new ExceedsMaxCapacityError();  }  Connect to Database room Table  Querry roomRecords table for roomType available during dateRange of checkIn checkOut;  if (#rowsReturned >= roomsNeeded) { //enough rooms available  Room r;  for each row returned {  int roomID = retrieve roomId from roomRecords;  String hotel = retrieve hotel from roomRecords;  int roomNumber = retrieve room# from roomRecords;  String roomType = retrieve type from roomRecords;  int basePrice = retrieve price from roomRecords;  r = new Room(roomID, hotel, roomNumber, roomType, basePrice);  //add Rooms to ArrayList  availableRooms.add(r);  }  }  else  throw unavailableReservationsError();  }//end method  Room lookUpRoom(int roomID) throws databaseLookUpError(){  //using roomID instead of # to account for possible multiple hotels  query database roomRecords with roomID  if (recordFound) {  Room r;  int roomID = retrieve roomId from roomRecords;  String hotel = retrieve hotel from roomRecords;  int roomNumber = retrieve room# from roomRecords;  String roomType = retrieve type from roomRecords;  double basePrice = retrieve price from roomRecords;  r = new Room(roomID, hotel, roomNumber, roomType, basePrice);  }  else  throw new databaseLookUpError();  }  //returns listOfAvailableRooms  ArrayListofRooms getAvailableRooms () {  return availableRooms;  }  //returns selected rooms  Int Array getSelectedRoomIDs {  Int Array selectedRoomIDs = new Int Array size roomsNeeded;  //Take first available rooms in array and automatically choose them for customer  for (j=0, j < roomsNeeded, j++) {  //store room IDs in an array to be returned  selectedRoomIDs[j] = availableRooms[j].getRoomID();  }  return selectedRoomIDs;  }  }//end class |

|  |
| --- |
| **class Room** {  //data  Final String HOTEL\_LOCATED; //the name of the hotel that the room is located in.  Final int ROOM\_NUMBER; //The room number of the given room.  Final int roomID;  Final double nightlyPrice;  String roomType; //A basic description of the room’s type    //methods  public Room(int id, String hotel, int roomNum, string description, double price) {  roomId = id;  HOTEL\_LOCATED = hotel;  ROOM\_NUMBER = roomNum;  roomType = description;  nightlyPrice = price;  }//end constructor  public toString() {  Display class variables;  }  }//end class |

Database Subsystem –

Each of the Records below will become tables that contain columns for data in an SQL Database.

CustomerRecords

CustomerID, firstName, lastName, street address, city, state, country, phone, email

ReservationRecords

ReservationID, CustomerID, RoomID(s), CheckInDate, CheckOutDate, Price, NumberOfGuests

RoomRecords

RoomID, Hotel, Type, Price, DatesBooked

UserRecords

EmpID, password, firstName, lastName

Risk and Mitigation

One of the main risks to the system is SQL injection. A SQL injection is essentially a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It allows an attacker to retrieve and view data that they normally wouldn’t have access to. The best methods for mitigating SQL injection are to use prepared statements (parameterized queries), use stored procedures, enforcing least privilege, and whitelisting validation.

Another risk we have to consider is a session timeout vulnerability. Insufficient session expiration is a security vulnerability that allows an attacker to reuse old credentials or session IDs that expose web applications or databases. The best mitigation is to invalidate a session after a predefined idle time has passed and provide users the means to invalidate their own sessions. Some other mitigations for this vulnerability is to avoid “infinite” session timeout, set session timeout to the minimal value possible, and use declarative definition of the session timeout in order to apply global timeout for all application sessions.

# Interface Control Document

## Revision Table

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Description | Contributor |
| 0 | 06/20/2020 | Created Document, Formatting, Added Sections, Started Inputting Data | Emmanuel Girin |
| 1 | 06/21/2020 | Updated Reservation Section | Emmanuel Girin |
| 2 | 06/22/2020 | Updated all sections. Ready for review | Emmanuel Girin |
| 3 | 06/23/2020 | Review of all Sections. Added Database Records | Leonardo Elias |
| 4 | 06/23/2020 | Review of all Sections | Marques Y. |
| 4 | 06/23/2020 | Review of all Sections | Jacob Valentine |

## Interface Control Document

This document describes the data flowing between the user and the input sub system as well as between the different subsystems. The data will be defined by type andrange of values.The headings below show the subsystem and the corresponding class in parentheses.Please refer to the project design document for the pseudocode outlining the classes.

## Outside User and Input(Hotel Reservation GUI)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| User | Input | empID | int | 0 to 2,147,483,647 | N/A |
| User | Input | password | String | maximum 25 characters, no spaces | N/A |
| User  Input | Input User | firstName | String | Maximum 50 characters | N/A |
| User  Input | Input User | lastName | String | Maximum 50 characters | N/A |
| User Input | Input User | street | String | Maximum 50 characters | N/A |
| User Input | Input User | city | String | Maximum 50 characters | N/A |
| User Input | Input User | state | String | Maximum 2 characters, no spaces | N/A |
| User Input | Input User | country | String | Maximum 50 characters | N/A |
| User Input | Input User | phone | String | Maximum 50 characters | N/A |
| User | Input | email | String | Maximum 50 characters | N/A |
| User Input | Input User | checkIn | Date | Present Day - 2022 | Present Day |
| User Input | Input User | checkOut | Date | Present Day - 2022 | Present Day |
| User Input | Input User | PreferedType | Integer | [0, 1, 2, 3, 4] | 0 |
| User Input | Input User | numRooms | Integer | 1 to 5 | 1 |
| User Input | Input User | numGuests | Integer | 1 to 10 | 1 |
| Input | User | loginMessage | String | “Error Incorrect Password”, “Incorrect Employee Id” | N/A |
| Input | User | customerLookUpMessage | String | “Failed to find customer. Exit or Try Again” | N/A |
| Input | User | reservationLookUpMessage | String | “Failed to find reservation. Exit or Try Again” | N/A |
| Input | User | price | String | 0.0 to 9999.00 | N/A |
| Input | User | reservationID | int | 0 to 2,147,483,647 | N/A |
| Input | User | databaseError | String | “Failed to connect to database. Try again later” | “Failed to connect to database. Try again later” |

## Input (Hotel Reservation GUI) and Logon (Login)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| Input | Login | empID | int | 0 to 2,147,483,647 | N/A |
| Input | Login | password | String | maximum 25 characters, no spaces | N/A |
| Logon | Input | ValidateCredentials | Boolean | true, false | false |
| Logon | Input | message | String | “”, “Error Incorrect Password”, “Incorrect Employee Id” | “” (blank) |

## Logon (Login) and Database (User Records)

The UserRecords database will contain first and last name of the employees, but this will never be used by the program only by staff to add or subtract the appropriate users from database. The creation of the database is left up to the one setting up the system.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| Login UserRecords | UserRecords Login | empID | int | 0 to 2,147,483,647 | N/A |
| Login UserRecords | UserRecords Login | password | String | Maximum 25 characters no spaces | N/A |

## Input (Hotel Reservation GUI) and Customer Subsystem (Customer Manager)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| Input | CustMgr | customerData | An Array of 9 Strings | 9 Strings, for limit of each string view Customer and Database interface | N/A |
| CustMgr | Input | custMgr.lookUp | Boolean | true, false | false |

## Customer (Customer Manager) and Customer Object (Customer)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| CustMgr | Customer  Object | CustomerData | An Array of 9 Strings | 9 Strings, for limit of each string view Customer and Database interface | N/A |
| CustMgr CustomerObj | CustomerObj CustMGr | customerID | int | -1 to 2,147,483,647 | N/A |
| Customer Object | CustMgr | firstName | String | Maximum 50 characters | “ “  (Blank String) |
| Customer Object | CustMgr | lastName | String | Maximum 50 characters | “ “  (Blank String) |
| Customer Object | CustMgr | streetAddress | String | Maximum 50 characters | “ “  (Blank String) |
| Customer Object | CustMgr | city | String | Maximum 50 characters | “ “  (Blank String) |
| Customer Object | CustMgr | state | String | Maximum 2 characters, no spaces | “ “  (Blank String) |
| Customer Object | CustMgr | country | String | Maximum 50 characters | “ “  (Blank String) |
| Customer Object | CustMgr | phone | String | Maximum 50 characters | “ “  (Blank String) |
| Customer Object | CustMgr | email | String | Maximum 50 characters | “ “  (Blank String) |

## Customer (Customer Manager) to Database (Customer Records)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| CustManager CustRecords | CustRecords  CustManager | CustomerID | Integer | -1 to 2,147,483,647 | N/A |
| CustManager CustRecords | CustRecords  CustManager | firstName | String | Maximum 50 characters | “ “  (Blank String) |
| CustManager CustRecords | CustRecords  CustManager | lastName | String | Maximum 50 characters | “ “  (Blank String) |
| CustManager CustRecords | CustRecords  CustManager | streetAddress | String | Maximum 50 characters | “ “  (Blank String) |
| CustManager CustRecords | CustRecords  CustManager | city | String | Maximum 50 characters | “ “  (Blank String) |
| CustManager CustRecords | CustRecords  CustManager | state | String | Maximum 2 characters, no spaces | “ “  (Blank String) |
| CustManager CustRecords | CustRecords  CustManager | country | String | Maximum 50 characters | “ “  (Blank String) |
| CustManager CustRecords | CustRecords  CustManager | phone | String | Maximum 50 characters | “ “  (Blank String) |
| CustManager CustRecords | CustRecords  CustManager | email | String | Maximum 80 characters | “ “  (Blank String) |

## Input (Hotel Reservation GUI) and Reservation Subsystem (Reservation Manager)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| Input | ReservMgr | checkIn | Date | Present Day - 2022 | Present Day |
| Input | ReservMgr | reservNumber | Integer | 1 to 2,147,483,647 | N/A |
| ReservMgr | Input/GUI | currentReservation | ReservationObject | N/A | null |
| Input | ReservMgr | checkOut | Date | Present Day - 2022 | Present Day |
| Input | ReservMgr | PreferedType | Integer | [0 to 5] | 0 |
| Input | ReservMgr | numGuests | Integer | 1-10 | 1 |
| Input | ReservMgr | numRooms | Integer | 1-5 | 1 |

### Input (Hotel Reservation GUI) and Reservation Object (Reservation)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| Input | Reservation Object | Reservation.toString() | String | Returns a formatted string from the data values of ReservationID, customerID, numOfGuests, totalCost, roomIDs, checkIn, checkOut. | N/A |

## Reservation (Reservation Manager) and Reservation Object (Reservation)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| ReservMgr | ReservObj | ReservationID | Integer | -1 to 2,147,483,647 | N/A |
| ReservMgr  ReservObj | ReservObj  ReservMgr | CustomerID | Integer | -1 to 2,147,483,647 | N/A |
| ReservMgr ReservObj | ReservObj  ReservMgr | RoomID | Integer | -1 to 2,147,483,647 | N/A |
| ReservMgr ReservObj | ReservObj  ReservMgr | CheckIn | Date | Present Day - 2022 | Present Day |
| ReservMgr ReservObj | ReservObj  ReservMgr | CheckOut | Date | Present Day - 2022 | Present Day |
| ReservMgr ReservObj | ReservObj  ReservMgr | Price | Double(Mgr) | 0.0 to 9999.00 | 0.0 |
| ReservMgr ReservObj | ReservObj  ReservMgr | NumGuests | Integer | 1 to 5 | 1 |

## Reservation (Reservation Manager) and Database (Reservation Records)

All the data flows both ways between the database and the class Reservation Manager. These subsystems will receive and send the data, so each row represents a flow of data in both directions.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| ResManager ResRecords | ResRecords ResManager | ReservationID | Integer | -1 to 2,147,483,647 | N/A |
| ResManager ResRecords | ResRecords ResManager | CustomerID | Integer | -1 to 2,147,483,647 | N/A |
| ResManager ResRecords | ResRecords ResManager | RoomID | Integer | -1 to 2,147,483,647 | N/A |
| ResManager ResRecords | ResRecords ResManager | CheckIn | Date | Present Day - 2022 | Present Day |
| ResManager ResRecords | ResRecords ResManager | CheckOut | Date | Present Day - 2022 | Present Day |
| ResManager ResRecords | ResRecords ResManager | Price | Double(Mgr) Float(Records) | 0.0 to 9999.00 | 0.0 |
| ResManager ResRecords | ResRecords ResManager | NumGuests | Integer | 1 to 10 | 1 |

## Input (Hotel Reservation GUI) and Available Room (Room Manager)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| Input | RoomMgr | checkIn | Date | Present Day - 2022 | Present Day |
| Input | RoomMgr | checkOut | Date | Present Day - 2022 | Present Day |
| Input | RoomMgr | preferredType | Integer | [0 to 4] | 0 |
| Input | RoomMgr | numRooms | Integer | 1 to 5 | 1 |
| RoomMgr | Input | SelectedRooms | Array of Room Objects | Memory Addresses of Room Objects | N/A |
| RoomMgr | Input | ExceedsMaxCapacity | Error | N/A | N/A |
| RoomMgr | Input | UnavailableReservations | Error | N/A | N/A |

## Available Room (Room Manager) and Room (Room Object)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| RoomMgr | RoomObj | Room ID | Int | -1 to 2,147,483,647 | N/A |
| RoomMgr | RoomObj | Room # | Int | 0 to 9999 | 0 |
| RoomMgr | RoomObj | Hotel Name | String | Maximum 50 characters | “” Blank String |
| RoomMgr | RoomObj | Type | String | Maximum 50 characters | ““ Blank String |
| RoomMgr | RoomObj | Price | Double (Mgr)  Float (Records) | 0.0 to 9999.00 | 0 |
| RoomMgr | RoomObj | Dates Booked | Date | Present Day - 2022 | Present Day |
| RoomObj | RoomMgr | selectedRoomIDs | Integer Array | Array of Memory Addresses pointing to Integers, each Integer can be 1 to 2,147,483,647 | N/A |

## Available Room (Room Manager) and Database (Room Records)

This table below describes the flow of data back and forth between the database and the java class Room Manager. It is important to note that the data for doubles in the SQL database will be stored as a float. The value of the type not the data object will be passed back and forth, so the types will automatically convert.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Name** | **Type** | **Value Range** | **Default Value** |
| Manager Records | Records  Manager | Room ID | Int | -1 to 2,147,483,647 | N/A |
| Manager Records | Records  Manager | Room # | Int | 0 to 9999 | 0 |
| Manager Records | Records  Manager | Hotel Name | String | Maximum 50 characters | “” Blank String |
| Manager Records | Records  Manager | Type | String | Maximum 50 characters | ““ Blank String |
| Manager Records | Records  Manager | Price | Double (Mgr)  Float (Records) | 0.0 to 9999.00 | 0 |
| Manager Records | Records  Manager | Dates Booked | Date | Present Day - 2022 | Present Day |

# Test Cases

## Revision Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Description** | **Contributor** |
| 0 | 06/23/2020 | Creating Document, Cover Page, Revision Table, and test case section. | Jacob Valentine |
| 1 | 06/23/2020 | Review of all Sections | Leonardo Elias |
| 2 | 06/23/2020 | Added test Case Descriptions and Expected Results | Jacob Valentine |
| 3 | 06/23/2020 | Finalized the Requirements Satisfied column | Leonardo Elias and Jacob Valentine |
| 4 | 07/12/2020 | Verified the actual results and pass/fail state of each test case. | Leonardo Elias and Emmanuel Girin |
| 5 | 7/12/2020 | Updated Requirements to show that 8 is filled | Emmanuel Girin |

Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case Number | Requirements Satisfied | Description | Expected Results | Actual Results | Pass/Failed |
| 1 | 12 | Start up the application. | The GUI should appear, and prompt the user for a login. | The GUI window appears and prompts the user to login. | Pass |
| 2 | 2,12 | When prompted for a login, the user will provide a valid employee ID number and associated password, and press enter. | The GUI application, upon verifying the login information, change to display the Customer manager screen. | The GUI application, upon verifying the login information, change to display the Customer manager screen. | pass |
| 3 | 2,12 | When prompted for a login, the user will provide an invalid Employee ID and any password, and press enter. | The GUI will display an error message saying that “Invalid Employee ID” before prompting the user for a valid login again. | The GUI displays an error message saying that “Invalid Employee ID” before prompting the user for a valid login again. | pass |
| 4 | 2,12 | When prompted for a login, the user will provide a valid Employee ID, but an invalid or non-associated password, and press enter. | The GUI will display an error message saying that “Invalid Password” before prompting the user for a valid login again. | The GUI displays an error message saying that “Invalid Password” before prompting the user for a valid login again. | pass |
| 5 | 1,3,12,14 | On the Customer Manager screen, select the “Create New Customer” option. When prompted for the new Customer’s information, enter valid information for all fields. | The system will add the customer information as a new entry into the customer records database, and the user will be given a message confirming the entry. | The system adds the customer information as a new entry into the customer records database, and the user receives a confirmation message | pass |
| 6 | 1,3,12,14 | On the Customer Manager screen, select the “Create New Customer” option. When prompted for the new Customer’s information, enter valid information for all fields except for payment information, which they will instead enter a non-numeric input into the credit card payment field. | The system will give the user an error message pointing out the invalid credit card input, and allow the user to resubmit the same form so long as they provide a new valid input for that field. | The system gives the user an error message pointing out the invalid credit card input, and allow the user to resubmit the same form. | pass |
| 7 | 1,4,15 | On the Customer Manager screen, the user will input the information of an existing customer entry to select a customer. | The system will check the customer records database for the selected customer entry, and return that customer’s information. This will set that customer as the current active customer, and the GUI will display the Reservation manager screen. | The system checks the customer records database for the selected customer entry, and return that customer’s information. This will set that customer as the current active customer, and the GUI displays the Reservation manager screen. | pass |
| 8 | 1,14,15,17 | On the Customer Manager screen, the user will input information not linked to any current customer record, and attempt to select that customer. | The system, upon finding no record for the customer, will display an error message to the user stating that no customer has been found matching that information, and the user will remain on the customer manager screen until proper information is entered. | The system, upon finding no record for the customer, displays an error message to the user stating that no customer has been found matching that information, and the user will remain on the customer manager screen until proper information is entered. | pass |
| 9 | 3, 14, 15 | The user inputs the information linked to an existing customer entry, and selects the “Modify Customer Account” option. When prompted for new information, input new valid information where needed. | The Customer entry of the CustomerRecords database is updated and the user is given a confirmation message. | The Customer entry of the CustomerRecords database is updated and the user receives a confirmation message. | pass |
| 10 | 9, 13, 17, 18 | On the reservation manager screen, have the user input a checkin date for a reservation linked to the active customer, and press “look up reservations” | The user is given a list of reservations that the customer is linked to for that given check-in day, alongside the information related to each reservation. If there is no reservation for that given day, the list will display a blank. | The user is given a list of reservations that the customer is linked to for that given check-in day, alongside the information related to each reservation. If there is no reservation for that given day, the list will display a blank. | pass |
| 11 | 9, 13, 17, 18 | On the reservation manager screen, have the user input a valid Reservation ID, and press “Select Reservation”. | The user is given the information for the associated reservation, and it is set as the actively selected reservation. | The user is given the information for the associated reservation, and it is set as the actively selected reservation. | pass |
| 12 | 9, 13, 17, 18 | On the reservation manager screen, have the user input an invalid Reservation ID not linked to any existing reservation, and press “Select Reservation” | The user will be presented an error message stating that no such reservation exists. | The user is presented with an error message stating that no such reservation exists. | pass |
| 13 | 13, 18, 19 | On the reservation manager screen, and with a reservation selected, the user presses the “delete reservation” button. When prompted on the “Are you sure” window, select “Yes”. | The system removes the reservation from the reservation database, and the user is presented a confirmation message confirming that the reservation has been canceled. | The system removes the reservation from the reservation database, and the user is presented a confirmation message confirming that the reservation has been canceled. | pass |
| 14 | 13, 18, 19 | On the reservation manager screen, and with a reservation selected, the user presses the “delete reservation” button. When prompted on the “Are you sure” window, select “No”. | The system remains as is, and the reservation remains both selected and in the reservation record. | The system remains as is, and the reservation remains both selected and in the reservation record. | pass |
| 15 | 9, 10, 11, 13, 17, 20, 22 | On the Reservation manager screen, the user inputs a checkin date, desired room style, and presses “Display available rooms”. | The system will return a message displaying a list of all matching rooms available for reservation on that checkin date, including their room ID and when their next reservation date will be if they have a future reservation. | The system will return a message displaying a list of all matching rooms available for reservation on that checkin date, including their room ID and when their next reservation date will be if they have a future reservation. | pass |
| 16 | 9, 13, 17, 20, 22, **8** | On the Reservation manager screen, the user inputs a valid room ID, checkin date, and checkout date, before pressing “Reserve Room”. | The new reservation is made and the user is shown a confirmation message. | The new reservation is made and the user is shown a confirmation message. | pass |
| 17 | 9, 13, 17, 20, 22, 8 | On the Reservation manager screen, the user inputs a an invalid room ID, as well as a valid checkin date and checkout date, before pressing “Reserve Room”. | The user is given an error message saying that the room ID could not be found. | The user is given an error message saying that the room ID could not be found. | pass |
| 18 | 3, 4, 5, 6, 7 | On startup, when prompted for a login, attempt to do so while the database is disconnected. | The user will be given an error message stating that the application could not access the database, and to try again later. | The user will be given an error message stating that the application could not access the database, and to try again later. | pass |
| 19 | 3, 12 | Close the application. | The application will disconnect with the Database and the GUI system will shut down. | The application disconnects from the database, the GUI shuts down, and the application will stop. | pass |

# User Guide

## Revision Table

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Description | Contributor |
| 0 | 07/11/2020 | Created User Guide Template | Marques Young |
| 1 | 07/12/2020 | Modified Users Guide | Jacob Valentine/Marques Young/ Leonardo Elias |

Hotel Reservation

User Guide

Table of Contents

Introduction

Purpose

Describing the System/System Specifications

Key Features

Inventory

Environment

System Operations

Installing, Starting and Stopping the System

Access Control

Accesing the system

Starting the system

Stopping/Suspending the system

Step by Step Instructions

Customer Management Screen

Reservation Management Screen

Errors/Malfunctions

Introduction

Purpose

The purpose of this document is to define the functionality that will be delivered by Group 1 for Hotel Reservations. This document serves as the sole reference for the scope of the system functionality to be delivered by Group 1 to its end user(s). This software will allow users to create a hotel reservation. The application is geared towards use by hotel front desk and sales employees. The software provides a solution for reserving available rooms on specific dates for customers.

Describing the System/System Specifications

Key Features

The Key Features of the Hotel Reservation system will be a Java Jar Application for users to reserve hotel rooms and the SQL database to store the user’s information for the purposes of identification and possible future reservations.

Inventory

The inventory for this web application is the SQL database that will be used to store a customer’s information to include their name, address, contact information, etc.

Environment

The Hotel Reservation system will use Windows Vista SP2 as the Operating System and Pentium 2 266 MHz processor or equivalent will be needed. The user will also need at least 3 MB of free RAM and 200 MB of disk space to operate the system efficiently. Keep in mind that 200 MB is the minimum, so more disk space may be required.

System Operations

* From the client-side the user will enter customer’s information which will consist of Personally Identifiable Information (PII) for the purpose of making a hotel reservations, creating new customers, modifying existing customers, and modifying existing reservations.
* The database will receive and store the information that the user has entered from the client-side for future queries such customers’ personal information, Room IDs, and Reservation iDs. If the user queries the database for any of the aforementioned information, it will be registered and displayed on the client side for the user's modification if need be.
* Each user will have his/her personal Employee ID and password to access the system, however, certain users will have more privileges than others to perform certain tasks within the system such as deleting reservations.

Installing, Starting and Stopping the System

Access Control

* As an employee/manager the user will be given an Employee ID and password to access both the client and server side of the system. The user will have access to the database in the instance that the user needs to modify a reservation or modify a customer’s information in relation to a reservation.
* As will be stated further in the User’s Guide, one of the access controls used in the system is least privilege; only managers are allowed to delete reservations, however, employees have the power to modify a reservation.

Installing the system

* The user will need to install the MySQL database which will be used to store customers’ information. MySQL is an open-source relational database management system (RDBMS) with a client-server model. MySQL creates a database for storing and manipulating data (such as customers’ information), defining the relationship of each table. Clients can make requests by typing specific SQL statements on MySQL. The server application will respond with the requested information and it will appear on the clients’ side. Make sure the Java Connector is installed with my SQL. Once MySQL is installed open MySQL workbench. Then, start the server and login to the admin account. Finally, on the top right corner there is a folder icon, click on it and import the TableData.sql file and run it to create the database, users, the tables, and the room data. There is some test data input in there for the customers benefit, which they can choose to remove. Since each hotel is different, a supervised installation of employee records and room records will need to be performed.

Accessing the system

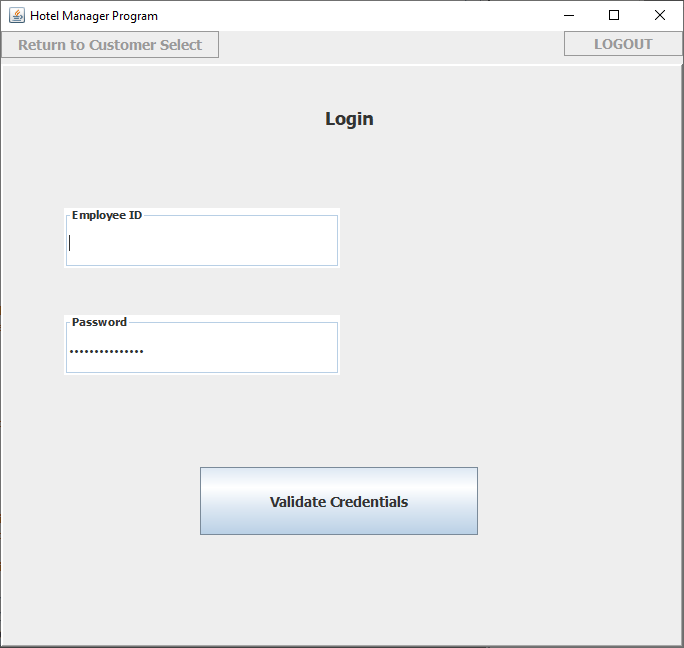
* After installing and configuring the proper MySQL systems, the user can access the HotelReservationSystem through the jar file included in the release package. Once the system is accessed the user will be prompted for his/her log on information which consists of the Employee ID and Password.

Stopping and Suspending the system

* To stop the system, the user simply has to click the X button on top of the main screen window. This will not only close the window but will also terminate the program safely. Information properly stored through the SQL will be saved in the database to be retrieved later.

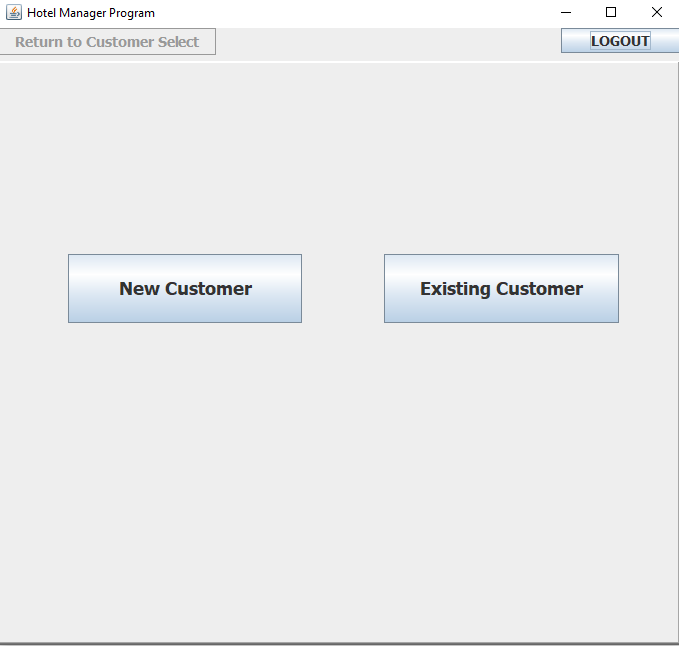
Step by Step Instructions

Login Screen



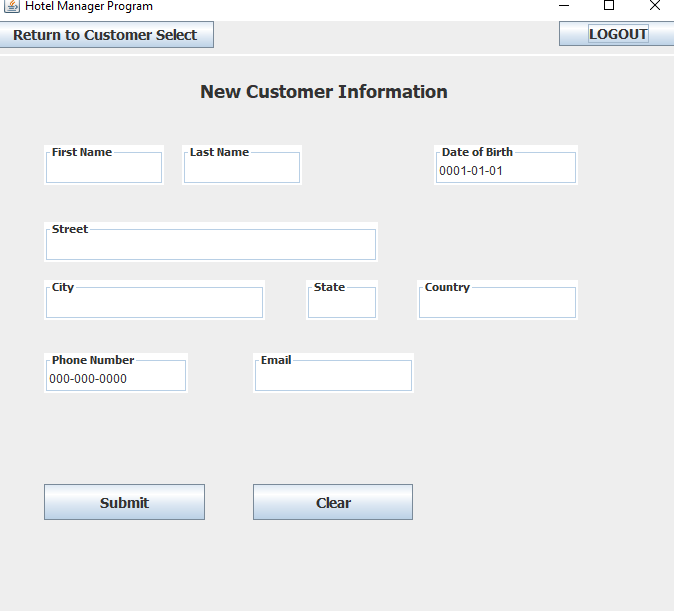
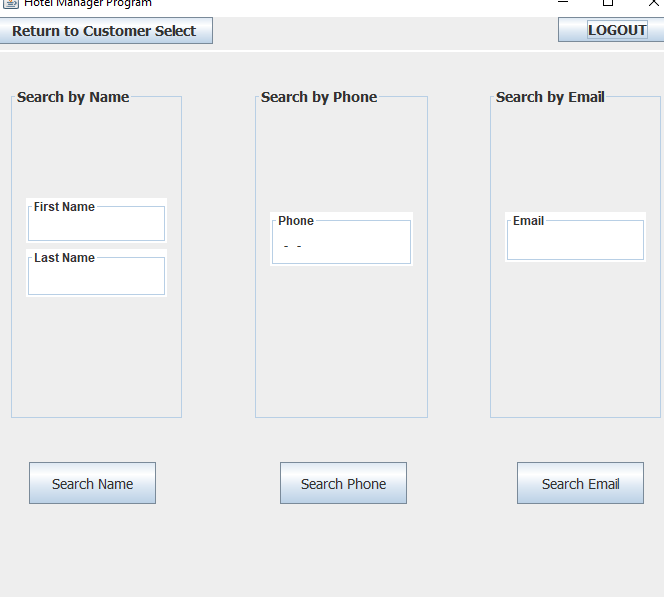
* Once the user (Employee/Manager) starts up the application, the GUI should appear and prompt the user for his/her logon information to include Employee ID and password.

Customer Management Screen

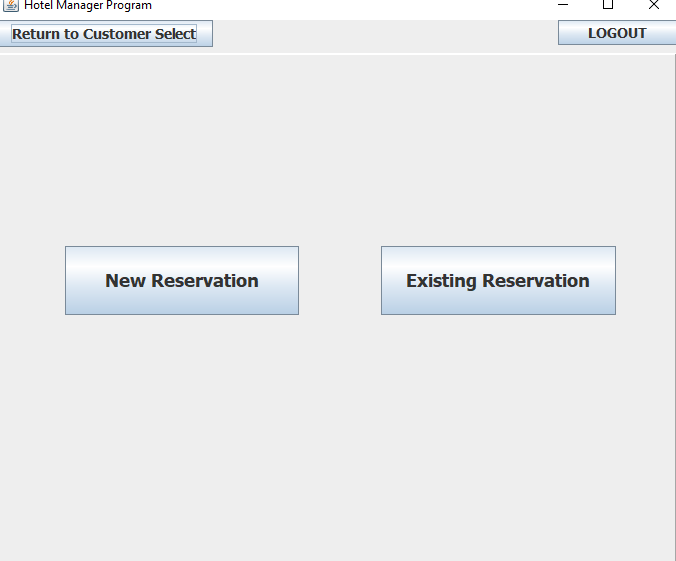


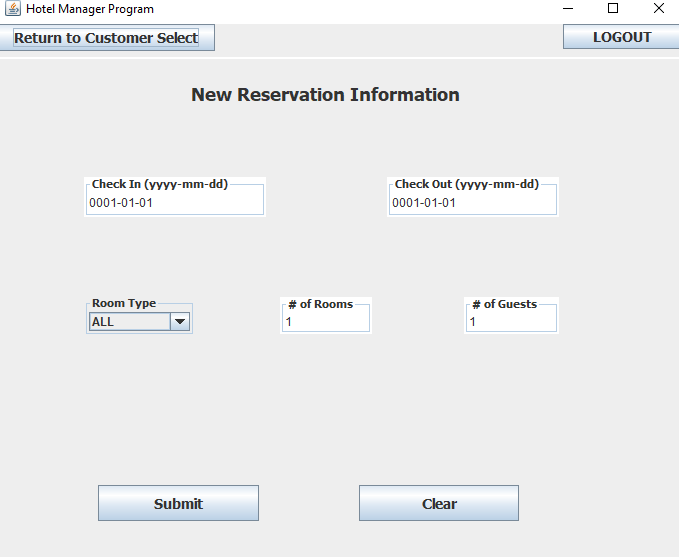
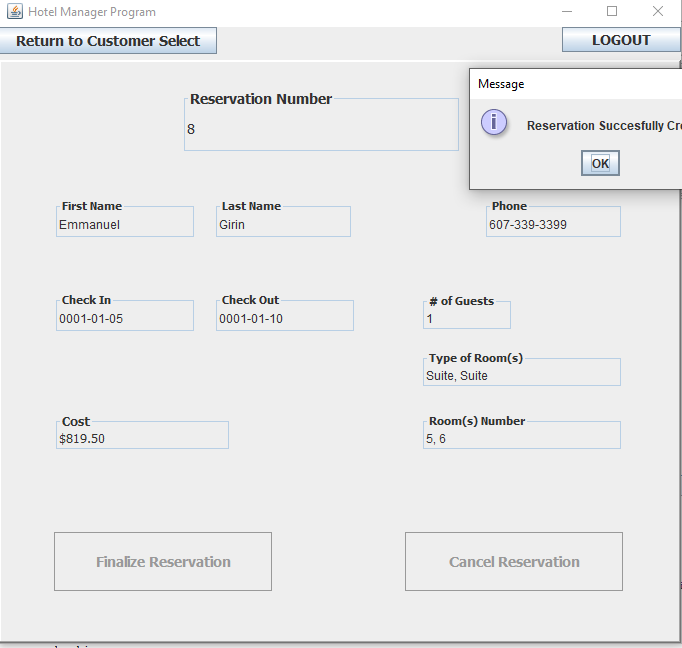
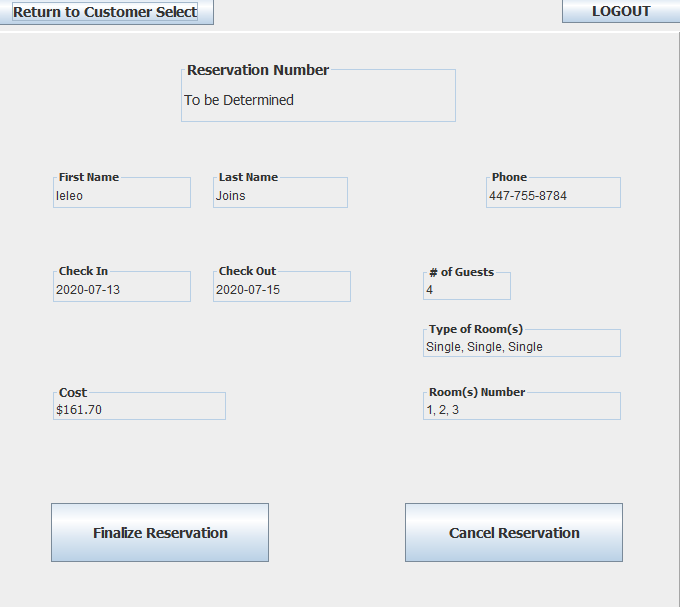
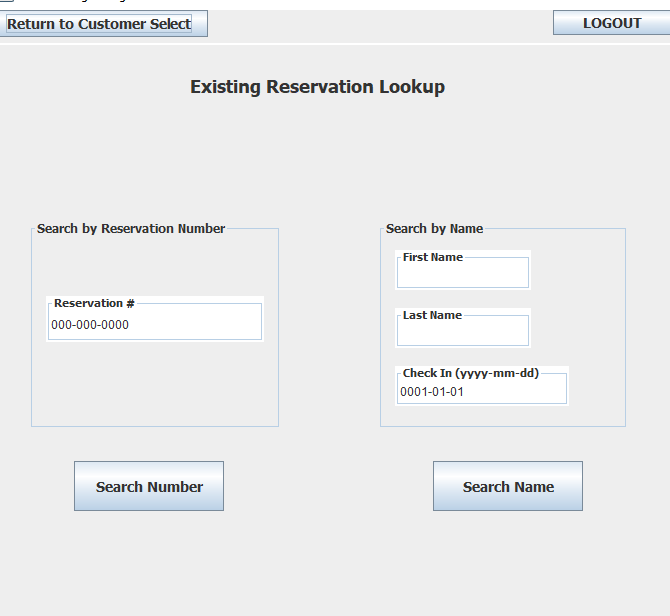
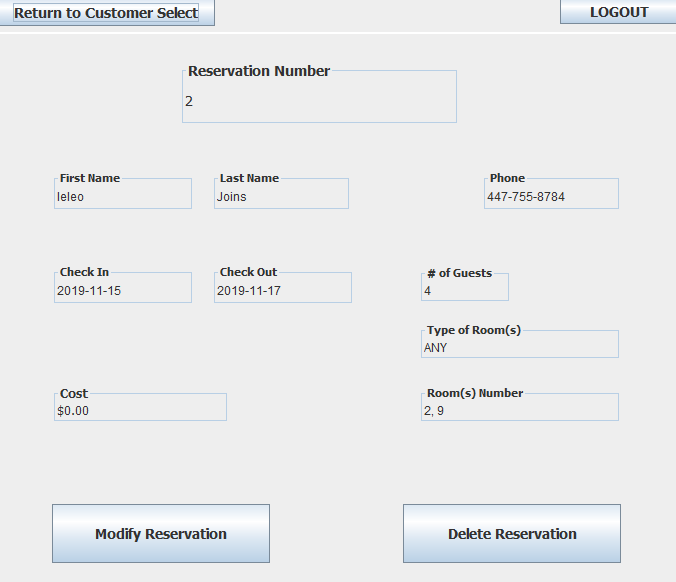
* The user (Employee/Manager) will then enter his/her logon information and be taken to the customer management screen where they will be able to Create New Customer.
* The user will enter the customer’s information which includes their name, address, date of birth, email, and contact information to be saved to the SQL database; once the information is saved to the database the user will get a confirmation that it has been done.
* If the user is looking for an existing customer he/she will enter the information of said customer and the system will return that particular customer’s information.

New Customer Existing Customer

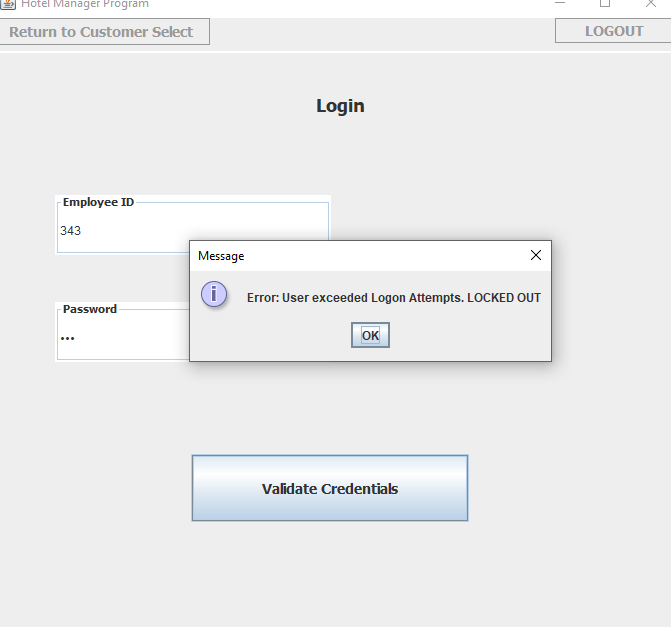
Reservation Management Screen

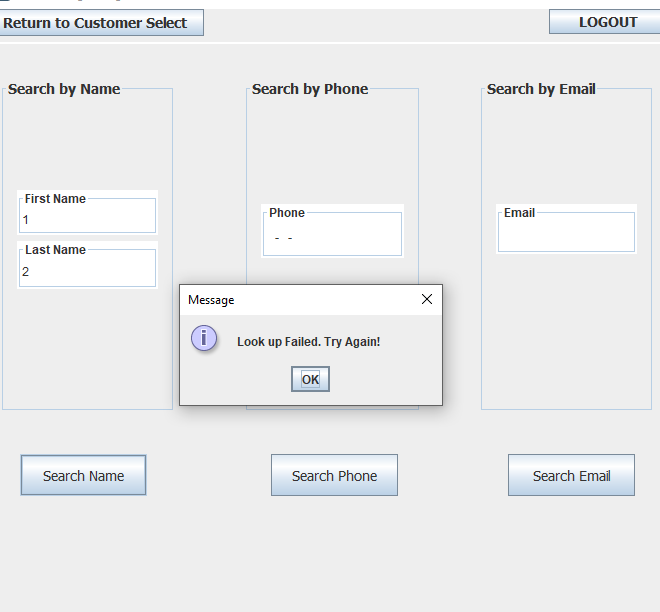
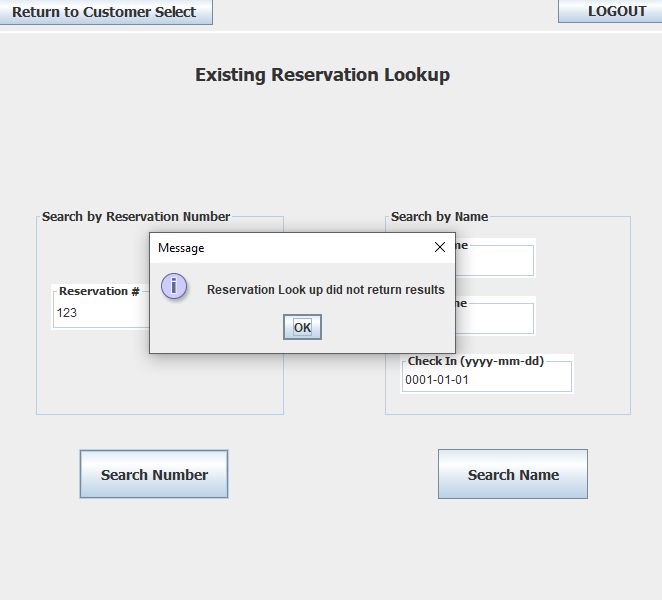


* On the reservation screen the user will be prompted to input a check-in date, room style, and will then press “Display Rooms”. The system will then display all rooms available for the particular check-in date; it will also display their room ID and their reservation date will be if they have a future reservation  
  
* Once the user inputs all required information (check-in date, Room ID, checkout date) he/she will be allowed to press the “Reserve Room” button. Once the reservation is completed the user will be shown a confirmation message.  
  
* From the Reservation Management Screen, the user is also able to delete the reservation. In order to delete the reservation, the user will press the “Delete Reservation” button.  
  
* The user will be displayed a message that asks if he/she is sure that they want to delete. Once pressed, the system will remove the reservation from the database and a confirmation will be sent stating that the reservation has been cancelled.  
  
* Bear in mind that only a manager, not any regular, is able to delete reservations.

Errors/Malfunctions

* If the user provides an invalid Employee ID (EMPID) the GUI will display an error message letting the user know that the logon attempt was invalid; the GUI will prompt the user to try to login again using the correct EMPID. The user will have 5 attempts before being locked out.



* If the user provides an invalid Password the GUI will display an error message letting the user know that the logon attempt was invalid; the GUI will prompt the user to try to login again using the correct password.
* While searching for an existing customer, if the system finds no information on the searched customer’s information, it will display an error message to the user indicating that no information was found on the searched user; the user will remain on the Customer Manager Screen until valid information is entered.  
  
* If the user enters an invalid Reservation ID the system will present an error message stating that no such reservation exists; the user may try again with a valid reservation ID.  
  
* If the user enters an invalid Room ID the system will present an error message stating that no such Room ID exists; the user may try again with a valid Room ID.
* If the database is disconnected when the user attempts to log on to the system, they will receive an error that lets the user know that the application could not access the database; the user will be prompted to try again.

Course Learnings

Creating the project over the course was an important learning experience. The importance of communication, planning, and documentation were all challenged and highlighted by this project. We ran into some complications submitting our deadline on time because the GUI was quite complicated and testing without the GUI was quite difficult. Our personal experiences reveal that more planning, and separating code into individual chunks that can be independently tested would have served us better. Instead we probably ended up doing what is unfortunately common in business scenarios, which is uncovering a lot of bugs at the end of the project and having to spend time fixing them. The experience of tying everything together ended up going smoothly. We would have liked to add the optional requirements of different SQL users and different types of employee logins, but these were optional additions that we couldn’t get to in time. Also, they require specific setup with different clients, so it would be best handled by a client’s own IT department.